IDA PAPER P-1266

IDAHEX VERSION 2

VOLUME II: Game Designer's Manual

Paul Olsen

May 1979

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Prepared for

Office of the Assistant Secretary of Defense
(Program Analysis & Evaluation)

INSTITUTE FOR DEFENSE ANALYSES
PROGRAM ANALYSIS DIVISION

IDA Log No. HQ 77-19281

The work reported in this document was conducted under Contract No. MDA903 79 C 0320 for the Department of Defense. The publication of this IDA Paper does not indicate endorsement by the Department of Defense nor should the contents be construed as reflecting the official position of that agency.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
IDAHEX VERSION 2
Vol II: Game Designer's Manual

5. TYPE OF REPORT & PERIOD COVERED
FINAL

6. PERFORMING ORG REPORT NUMBER
IDA PAPER P-1266

7. AUTHOR(s)
Paul Oslen

8. CONTRACT OR GRANT NUMBER(s)
DAHC 15 73 C 0200

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Institute for Defense Analyses
400 Army-Navy Drive
Arlington, VA 22202

10. PROGRAM ELEMENT PROJECT TASK AREA & WORK UNIT NUMBERS
Task PA&E 106

11. CONTROLLING OFFICE NAME AND ADDRESS
Office of the Assistant Secretary of Defense
(Program Analysis & Evaluation)
Pentagon, Washington, DC 20301

12. REPORT DATE
May 1979

13. NUMBER OF PAGES
198

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Office of the Assistant Secretary of Defense
(Program Analysis & Evaluation)
Pentagon
Washington, DC 20301

15. SECURITY CLASS (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Land Warfare, Ground Combat, Simulation, Interactive Model, War Game, Computer-Assisted War Game, Ground Forces, Ampnibious Landings, Airborne Operations, Maneuver

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
IDAHEX is an interactive computer model of two-sided conventional land warfare. It keeps the players informed of the situation and accepts their instructions to their forces. Units can move by land, sea, or air. A unit's movement rate is variable, depending upon its posture, the conditions of its movement, and the adequacy of transport. Attrition in engagements is assessed by a heterogeneous Lanchester square process. Indirect supporting fire and direct air support

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Item 20 continued

can be played. There is a simplified representation of air attacks on lines of communication and of engineering activities, including bridge building and mine laying. Supplies consumption can be assessed and logistics can be played. The model recognizes severed lines of retreat and lines of supply and imposes appropriate consequences. The documentation consists of three volumes: (1) A Guide for Potential Users; (2) Game Designer's Manual; (3) Player's Manual.

IDA PAPER P-1266

IDAHEX VERSION 2

VOLUME II: Game Designer's Manual

Paul Olsen

May 1979

INSTITUTE FOR DEFENSE ANALYSES
PROGRAM ANALYSIS DIVISION
400 Army-Navy Drive, Arlington, Virginia 22202

Contract No. DAHC 15 73 C 0200
Task PA&E 106

PREFACE

IDAHEX is a computerized model of conventional land warfare at the theater level. Its documentation consists of:

Volume 1: *A Guide for Potential Users*
Volume 2: *Game Designer's Manual*
Volume 3: *Player's Manual*

Volume 1 outlines the model's fundamental characteristics. Volume 2 (this volume) comprehensively describes the model and its data base. Volume 3 contains enough information for someone with a modest knowledge of land warfare to play an IDAHEX game under the general supervision of the game designer. It outlines the entire model, identifies information the game designer should give the players, and describes IDAHEX as a war game from the players' perspective.

Comments and inquiries are welcomed. They should be directed to the author (commercial telephone 202-697-0584, autovon 227-0584).

CONTENTS

FIGURES

TABLES

1. UNDERSTANDING THE MANUAL

IDAHEX is a model of warfare. The model has been implemented as a computer program, written primarily in FORTRAN. Usually, no distinction is drawn between the model and the program; this manual refers to both as "IDAHEX".

As a model, IDAHEX imposes some structure: there are exactly two sides in the conflict, "Red" and "Blue"; each side's force consists of individual "battle units"; the postures that a battle unit can assume are organized into sequenced classes; the area in which the Red and Blue forces move and fight, termed the "area of war", is approximately rectangular and is partitioned into regular hexagons; each battle unit's location is identified with a hexagon. Within this structure the "game designer" creates a game. He specifies the compositions of the Red and Blue forces, the resources held by each battle unit, the postures battle units can assume, the battle units' mobility, their resources' effectiveness in combat, the terrain in the area of war, and the size of the hexagons. Loosely speaking, the model is a war game whose rules are parameterized; the game designer sets the parameters, turning a general structure into a specific game. As a tool for designing war games, IDAHEX is valuable because:

(1) it is systematic, and therefore protects against design errors and omissions;

(2) it incorporates reasonably sophisticated procedures for assessing movement, combat, supplies consumption, and LOC interdiction, obviating the time-consuming alternatives of writing *ad hoc* computer programs and making the assessments manually;

(3) it contains extensive logic for handling the consequences of maneuver.

The last point deserves emphasis. If a model plays maneuver at all--i.e., if battle units can move in more than one dimension--engagements may start and stop, battle units may be attacked from multiple directions, attackers may be attacked from the flank or rear, and enemy units may meet on the march. IDAHEX can handle these events. Without that capability, the game

designer would have to rely on a control team to make *ad hoc* judgments or would have to write a web of rules.

This manual describes the IDAHEX model and shows how to use IDAHEX to design a war game. The Glossary briefly defines all variables input by the game designer as well as key variables and functions used internally by the IDAHEX computer program. Detailed information on almost any variable or function in the Glossary can be found through the Index. Some variables and functions--easily identified because their names appear in brackets--may not actually exist in the IDAHEX program, but may be regarded just as any other variables and functions. They have precise analogs in the program, analogs that may be pedagogically inconvenient because of special coding to conserve storage. A variable whose name begins with a capital letter may not correspond to any program variable; it is only a pedagogical device. If a variable's value is set by inputs from the game designer--the "game design data"--the variable's name is italicized or underlined. In some cases, a variable's value is set by the game design data but may be altered later by IDAHEX; the altered variable is identified by the same name without italics or underlining. Examples:

(1) The array *katk* is set by the game design data, but IDAHEX almost immediately redefines it by multiplying each element by *tframe*. The resulting variable is named "katk" to distinguish it.

(2) The vector of battle units' locations, *buloc*, is initially set by the design data. But units' locations may change during the game. The vector variable containing updated unit locations is named "buloc".

A variable's name is never italicized or underlined simply because its value is derived from game design data: ultimately, every variable's value is determined by the game design data and the players' inputs.

The reader may be unaccustomed to variables' names containing lower-case letters. The IDAHEX documentation uses program variables' names as they appear in the MULTICS version of the IDAHEX computer program. In MULTICS FORTRAN and PL/I, the lower-case letters constitute the primary alphabet, of which the upper-case letters are an extension. Changing every lower-case letter in the IDAHEX source program to upper-case produces a logically equivalent program.

Unless the contrary is affirmed, a variable determining the number of elements in a set may be 0. For example, the game design data fix the value of *nss*(1), the number of types of Red

supplies. Such size parameters let the game designer choose from a spectrum of complexity. At one end he can play several types of weapons, several types of transport, several types of supplies, and several types of personnel on a side. At the other end, he can play just one resource--an abstract index of strength--on a side.

IDAHEX distinguishes three roles for its user or users: the game designer, who provides the inputs that specify the game (the game design data); the Red player, who commands the Red force; and the Blue player, who commands the Blue force. If used in an interactive mode, IDAHEX gets the game design data from one file--ordinarily associated with the card reader, a tape data set, or a disc data set--and gets the players' inputs from one or two terminals. For maximum clarity, the documentation is written as though IDAHEX is used interactively.

The term "unit" means "battle unit" unless the context in which it is used indicates otherwise. The phrase "a Red type i resource", or "a Red resource of type i", means "a unit-quantity of Red resources of Red resource type i". (Here, "unit" means "unit of measure", not "battle unit".) Likewise for Blue. An element of a vector or a (two-dimensional) matrix may be identified by use of parenthesized arguments instead of subscripts:

$$x(i) \text{ means } x_i,$$

$$a(i,j) \text{ means } a_{ij}.$$

The variable a(i,*) is row i of the matrix a. The variable a(*,j) is column j of the matrix a. These may also be written as

$$a_{i*} \quad \text{and} \quad a_{*j}.$$

The symbol "ε", when used syntactically, means "in", "belongs to", or "is a member of". Example: if C is a set, "u ε C" means "u is a member of C". The same symbol with a line through it ("∉") means "not in", "does not belong to", or "is not a member of". If y and z are scalar variables, y*z denotes their product, and y**z denotes y raised to the power z.

2. THE ELEMENTS OF PLAY

This section explains how IDAHEX structures the area of war, the resources, and changes in unit postures and locations.

2.1 THE AREA OF WAR

The game board is termed the "area of war"--the area in which the forces exist. It is partitioned into congruent, regular hexagons, as Figure 2.1 illustrates. The hexagons are termed "cells". A cell's *depth* is defined as the distance from one side to the side directly opposite it; this distance equals the distance from a cell's center to any adjacent cell's center. (See Figure 2.2.) The variable *depth* is fixed by the game design data. The cells are always arranged in ranks and numbered as Figure 2.1 illustrates. The number of cells in the first (the leftmost) rank, *nrank1*, is fixed by the design data. (It is 8 in Figure 2.1.) The next rank always has one less cell. The number of ranks is jointly determined by *nrank1* and *ncells*, the number of cells in the area of war. The highest numbered cell in a rank (in Figure 2.1, the bottom cell in each rank) is always inactive, regardless of the game design data.

A cell's "terrain type" symbolizes the natural and man-made features in the cell that would affect cross-country movement, combat, or vulnerability to air strikes. Examples: clear, hilly, built-up, fortified. Since only one terrain type corresponds to a cell, there is an implicit assumption that the conditions within a cell are uniform. The terrain type of cell i is coded in the game design variable [*terrain*](i) as a non-negative integer. It is 0 if and only if cell i is inactive. Making a cell inactive reduces IDAHEX's computer storage requirement.

Information on cross-country movement rates between two adjacent cells derives from their terrain types. Complete information on trafficability must also include characterizations of the road and railroad communications between the cells. In a theater-level game, the cells would normally be so large that more than one road or more than one railroad might run between two adjacent cells. A characterization of road or rail communications between two adjacent cells should recognize redundancy--the presence of multiple roads or multiple railroads

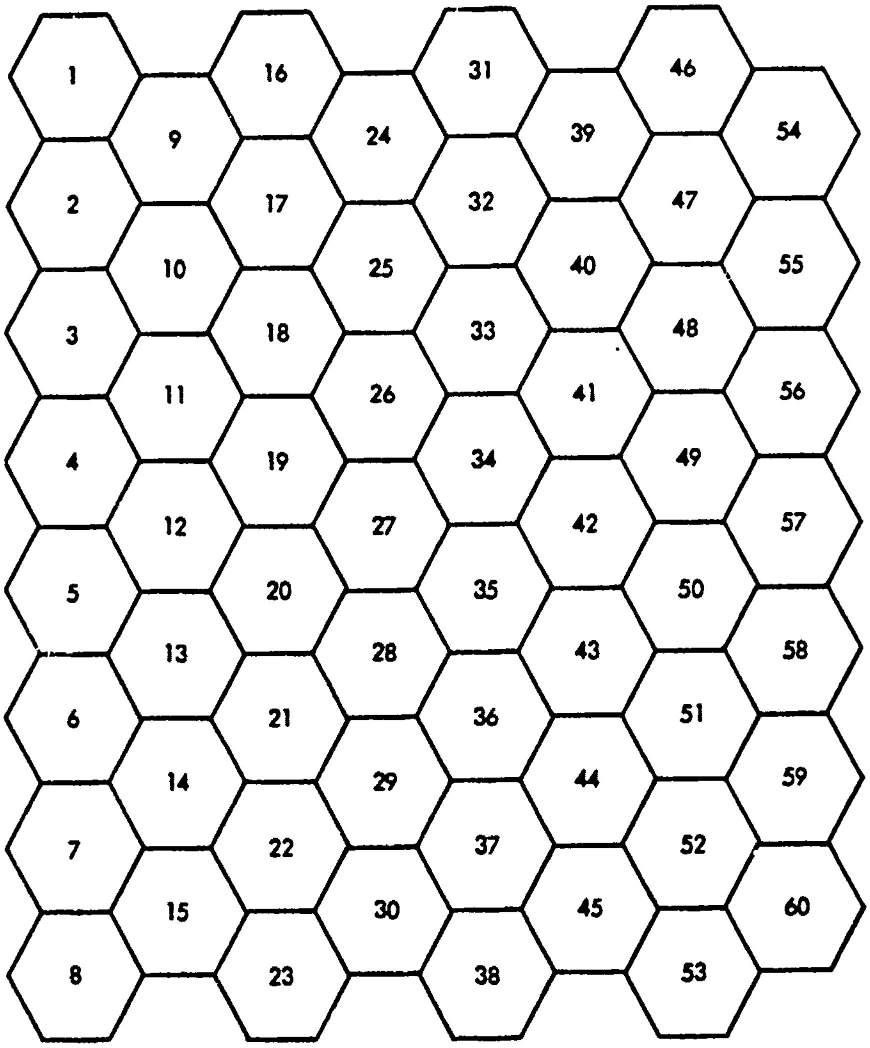


Figure 2.1. EXAMPLE OF AREA OF WAR

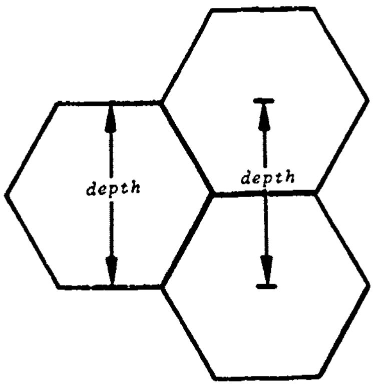


Figure 2.2. ILLUSTRATION OF CELL DEPTH

running directly from one cell to the other--as well as the usual distinctions such as paved or unpaved, standard gauge or narrow gauge. If cell i and cell j are adjacent, the game design datum [*basic_road*](i,j) is the basic characterization of the road link between them, coded as a nonnegative integer; [*basic_rail*](i,j) is the basic characterization of the rail link between them, coded as a nonnegative integer. If [*basic_road*](i,j) = 0 or [*basic_rail*](i,j) = 0, it means that there are no (direct) road communications or no (direct) rail communications between the cells. Note that "road" should be interpreted broadly: a road is defined as any path whose trafficability is significantly better than the general surroundings'.

Movement between two adjacent cells, and also combat between a force located in one and a force located in the other, may be affected by a "barrier" between the cells. Barriers include rivers, ridges, fortified lines, and, in general, any natural or man-made obstacle that significantly affects movement or attack across it. If cell i and cell j are adjacent, [basic_barrier](i,j) is the basic characterization of the type of barrier between them, coded as a nonnegative integer; [basic_barrier](i,j) = 0 signifies that there is no barrier. In reality, there may be several obstacles between the cells; [basic_barrier](i,j) is a single number characterizing them as a whole.

The game design data fix [*basic_barrier*]. At the start of the game, IDAHEX sets

[basic_barrier](i,j) = [*basic_barrier*](i,j)

for every pair of adjacent cells, i and j. In contrast to [*terrain*], [*basic_road*], and [*basic_rail*], [basic_barrier] may change during the game due to engineer operations (as Section 8 explains).

Necessarily,

[*basic_road*](i,j) = [*basic_road*](j,i),
[*basic_rail*](i,j) = [*basic_rail*](j,i),
[basic_barrier](i,j) = [basic_barrier](j,i)

for any adjacent cells, i and j. If cell i or cell j is inactive, then IDAHEX sets

[*basic_road*](i,j) = 0,
[*basic_rail*](i,j) = 0,
[basic_barrier](i,j) = 0,

regardless of the input data.

IDAHEX permits detailed terrain types, basic road-link types, and basic barrier types. The maximum permitted value of [*terrain*](i) is nenvraw, the maximum permitted value of [*basic_road*](i,j) is nrdraw, the maximum permitted value of [*basic_rail*](i,j) is nrrraw, and the maximum permitted value of [*basic_barrier*](i,j) is nbarrar. The bounds nenvraw, nrdraw, nrraw, and nbarraw are fixed by the entry point cgcm. This level of detail could not be accommodated in the assessments of movement and combat without imposing unreasonably large computer storage requirements. Therefore, IDAHEX uses a collection of mappings to transform the terrain type into the "environment type", the basic road-link type into the "road-link type", the basic rail-link type into the "rail-link type", and the basic barrier type into the "barrier type". The mappings are defined by the game design data and may be redefined during the war; as a result, the mappings provide a way of altering the environment, road, rail, and barrier types to reflect changes in weather and even the change from day to night and back.

A cell's "environment" is the complex of physical conditions in the cell that affect cross-country movement, combat or vulnerability to air strikes. The environment in cell i is coded in [environment](i) as a nonnegative integer. By definition,

$$[\text{environment}](i) = \begin{cases} \mathit{mapter}([\mathit{terrain}](i)) & \text{if } [\text{terrain}](i) > 0, \\ 0 \text{ otherwise.} \end{cases}$$

The type of road link between two adjacent cells, i and j, is coded in [road](i,j) as a nonnegative integer; [road](i,j) = 0 signifies that no road link exists. By definition,

$$[\text{road}](i,j) = \begin{cases} \mathit{maprd}(\text{brd}) & \text{if brd} > 0, \\ 0 & \text{if brd} \le 0, \end{cases}$$

where brd = [*basic_road*](i,j).

The types of rail link between two adjacent cells, i and j, is coded in [rail](i,j) as a nonnegative integer; [rail](i,j) = 0 signifies that no rail link exists. By definition,

$$[\text{rail}](i,j) = \begin{cases} \mathit{maprr}(\text{brr}) & \text{if brr} > 0, \\ 0 & \text{if brr} \le 9, \end{cases}$$

where brr = [*basic_rail*](i,j).

Some barriers that have approximately the same direct effect on combat (in terms of their degradation of the attackers' effectiveness) may have very different effects on unopposed movement, and *vice versa*. Therefore, it is helpful to have

completely independent methods of aggregating basic barrier types into barrier types used for combat--"attack barrier types"--and aggregating basic barrier types into barrier types used for movement--"movement barrier types". The game design variable *mapab* is used to map basic barrier types into attack barrier types; the variable *mapmb* is used to map basic barrier types into movement barrier types.

The type of movement barrier between two adjacent cells, i and j, is given by the function value movebar(i,j), a non-negative integer; movebar(i,j) = 0 signifies that no movement barrier exists (but an attack barrier may exist). By definition,

$$\text{movebar}(i,j) = \begin{cases} \mathit{mapmb}(\text{bb}) & \text{if bb} > 0, \\ 0 & \text{if bb} \le 0. \end{cases}$$

Normally, bb = [basic_barrier](i,j). But as Section 9 explains, ground forces and air power may have intensified the barrier, making a different value of bb appropriate. For example, if the value of [basic_barrier](i,j) signified a bridged river, but the bridges had been rendered unusable during the game, a value of bb signifying an unbridged river would be appropriate.

A movement barrier between adjacent cells, i and j, has the same effect on movement from cell i to cell j as on movement from cell j to cell i. In contrast, a barrier's effect on attack may differ according to the direction of the attack, and therefore it might be appropriate to use a different attack barrier type for an attack from cell i toward cell j than for attack from cell j toward cell i. There are six directions in the area of war; Figure 2.3, depicting a subset of the area of war in Figure 2.1, shows how they are numbered from 1 through 6 for identification. Suppose cell j lies in direction $k (1 \le k \le 6)$ from cell i. The type of attack barrier confronting an attack from cell i toward cell j is given by the function value atkbar(i,j), a non-negative integer; atkbar(i,j) = 0 signifies that no attack barrier exists (but a movement barrier may exist).

Define bb as in the preceding paragraph. Then, by definition,

$$\text{atkbar}(i,j) = \begin{cases} \mathit{dirab}(\mathit{mapab}(\text{bb}),k); & \text{bb} > 0,\ \mathit{mapab}(\text{bb}) > 0 \\ 0; & \text{otherwise.} \end{cases}$$

Thus, if bb = 0, or if bb > 0 but *mapab*(bb) = 0, then no attack barrier exists. Otherwise, *mapab*(bb) is, in a sense, a first estimate of the attack barrier type. The game design datum *dirab*(m,k) tells what attack barrier type to use if the first estimate (made without considering the direction of the attack) is m and the direction of attack is k. Note that *dirab*(m,k) may equal 0, implying no attack barrier.

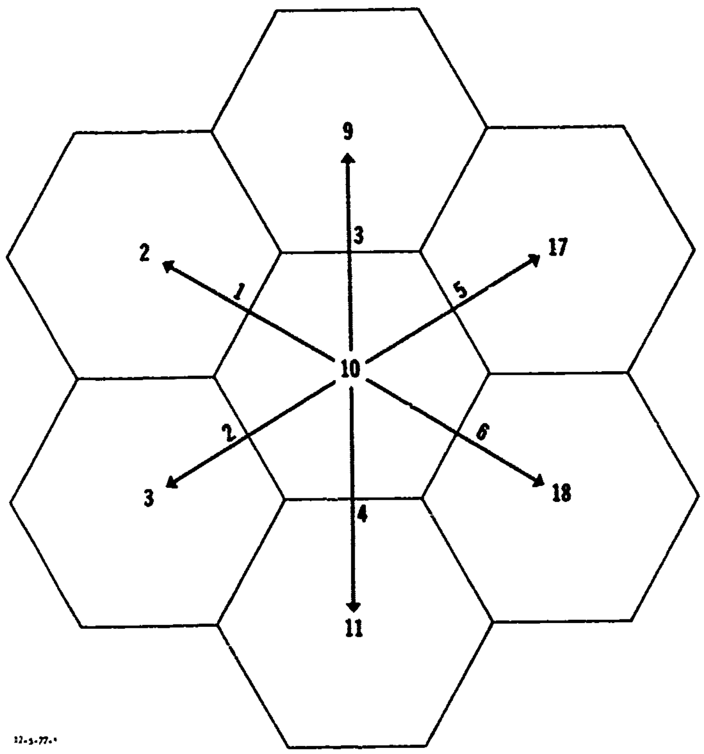


Figure 2-3. THE SIX DIRECTIONS IN AN AREA OF WAR

Initially,

$$\begin{aligned} mapter(\mathrm{i}) &= \mathrm{i}, \\ maprd(\mathrm{i}) &= \mathrm{i}, \\ maprr(\mathrm{i}) &= \mathrm{i}, \\ mapab(\mathrm{i}) &= \mathrm{i}, \\ mapmb(\mathrm{i}) &= \mathrm{i} \end{aligned}$$

for any i. At the start of every cycle, including the first, the game design data can modify the maps, which then remain fixed unless and until the design data modify them again.[1] (To modify the maps is to change one or more elements of the vectors *mapter, maprd, maprr, mapab,* and *mapmb*.) IDAHEX offers to advise the players of modifications.

The game design datum *nenv* should equal the maximum possible value of [environment](i) for any i; *nroad, nrail, nmb,* and *nab* should equal the maximum possible values of [road](i,j),

---

[1]A "cycle" is a subdivision of game time. It is defined in Section 2.3.

[rail](i,j), movebar(i,j), and atkbar(i,j) for any pair of adjacent active cells, i and j, respectively. To reduce the IDAHEX computer program's storage requirements, *nenv, nroad, nrail, nmb,* and *nab* should be kept small in value.

2.2 THE FORCES

There are two forces, Red and Blue. Each force consists of indivisible "battle units", often called simply "units". The game designer assigns each unit a unique number by which IDAHEX identifies it. A unit's number must be a positive integer. The number assigned to any Red unit must be less than any number assigned to a Blue unit, but units need not be numbered consecutively. Each unit has a "name"--a character string--and a type. The complete set of unit types for a particular game might be:

1. Red motorized rifle division
2. Blue tank division
3. Red tank battalion
4. Red tank division
5. Red transport unit
6. Blue transport unit
7. Blue infantry division

Each unit type is identified by a positive integer. A unit's type is stored in the vector *butype;* in terms of the example above, if unit 45 is a Red tank division, then *butype*(45) = 4. Units of the same type must belong to the same side.

2.2.1 Battle Unit Status

A battle unit's "status" is described by its location, posture, and objective. Each battle unit is located in exactly one cell. The unit's location can not be fixed more precisely: the unit is never said to be, for example, 3 km east of the cell's center. Its location *is* the cell. Several units may have the same location, even if they belong to different sides.

At any moment of the game, each battle unit is in one of 6 "posture classes":

- -1. destroyed
- 0. inactive
- 1. hold
- 2. disengagement
- 3. movement
- 4. attack

A unit in posture class 2 is trying to break contact with any enemy units it may be fighting, as the first step in changing location. Its "objective" is the cell toward which it is disengaging. A unit in posture class 3 is moving from its location to another cell, its objective. Ordinarily, a unit in posture class 4 is trying to enter a new location, which may or may not contain enemy units, but in some cases it is trying to revert from posture class 2, 3, or 4 to posture class 1 without changing location. In the former instance, its objective is the cell it seeks to enter; in the latter, its objective is just its present location.

Posture class 1 embraces all remaining activities as well as simple idleness. In particular, a unit in posture class 1 is not in the process of changing location. It may or may not be engaged. Its objective is, by convention, its location.

A unit in posture class -1 or 0 is said to be "inactive". (Inversely, a unit in a positive posture class is said to be "active".) An inactive unit does not exist from the perspectives of other units. It can not move; it can not attack, nor can it be attacked. A unit in posture class -1 is a special kind of inactive unit: it was de-activated to represent its destruction, usually as a result of suffering intolerably high losses. A unit in posture class 0 is ordinarily a reinforcement or a package of replacements. It may become active (enter a positive posture class) later in the war. Its location is the cell where it is expected to enter the area of war if it becomes active, but while it remains inactive, it has no effect on enemy units passing through its location.

When a unit, say unit j, enters a new posture class, its "virtual time of entry", tentry(j), is updated. Normally, tentry(j) is set to the exact time at which unit j enters the new posture class, but in special situations it may be set to a later time. At the start of the game, tentry = *tentry*. The game design datum *tentry*(j) is most simply defined as the time at which unit j entered the cell where it is located at the start of the game. For example, if the game starts at time 0, if time is measured in days, and if unit 45 assumed its starting location 30 days prior to the starting time, then *tentry*(j) should be -30.0.

Each positive posture class consists of at least one posture and no more than 10 postures. Posture class -1 consists of just one posture, numbered -10. The postures in posture class 0 are numbered 0 through 9, but IDAHEX does not distinguish among the postures in posture class 0. The postures in posture classes 1 through 4 are numbered as follows:

10-19 hold
20-29 disengagement
30-39 movement
40-49 attack

(Notice that [floor](p/10) is the posture class to which posture p belongs.[1]) There might, for example, be two different movement postures, representing surface movement and airborne movement. There might be several different attack postures, representing different degrees of willingness to trade casualties for space. Table 2.1 presents alternative ways of describing a unit's posture class. The game design datum *npost*(i) fixes the number of postures in posture class i ($1 \leq i \leq 4$). There must be one posture numbered 10, one numbered 20, one numbered 30, and one numbered 40. These are the standard hold, disengagement, movement, and attack postures; if IDAHEX knows a unit's posture class but has insufficient information to determine the posture, it assumes the standard posture in the posture class.

Table 2.1. EQUIVALENT DESCRIPTIONS OF POSTURE CLASS

in posture class 1; in a hold posture;	holding
in posture class 2; in a disengagement posture;	disengaging
in posture class 3; in a movement posture;	moving
in posture class 4; in an attack posture;	attacking

The postures within a posture class need not be numbered consecutively, but doing so may reduce storage requirements. Some postures within posture class 3 may represent road or cross-country movement whereas others may represent rail, air, or sea movement. Numbering road/cross-country movement first may reduce storage requirements.

2.2.2 Battle Unit Resources

The types of resources each side has are arranged in a list. For example, the list of Red resource types might be:

1. tanks
2. small arms and APCs
3. artillery

[1]See the Glossary for the definition of the function [floor].

4. SAMs and AAA
5. trucks
6. ammunition
7. fuel and other consumables
8. tank crewman
9. other personnel

The Blue list might be:

1. small arms and APCs
2. artillery
3. tanks
4. trucks
5. supplies
6. personnel

There is no correspondence between Red resource types and Blue resource types: in the example, Red type 3 resources are artillery while Blue type 3 resources are tanks, and Red has SAMs and AAA while Blue has none. The resource types must be listed in the following order:

ground-to-ground weapons
ground-to-air weapons
transport
supplies
personnel

The rubric "ground-to-ground weapon" applies to any weapon that belongs to a battle unit and is potentially capable of inflicting materiel losses on the enemy. It is reasonable to include attack helicopters in this category: they are usually attached to ground forces, they have short range, and their lethality and vulnerability depend on the environment of the cell in which they are operating. The preceding five resource categories are combined to form larger categories:

materiel	ground-to-ground weapons	weapons	equipment
materiel	ground-to-air weapons	weapons	equipment
materiel	transport		equipment
materiel	supplies		support
	personnel		support

The preceding categories induce sublists in each side's list of resource types. In the example above, the list of Red ground-to-ground weapons is:

1. tanks
2. small arms and APCs
3. artillery

The list of Red weapons is:

1. tanks
2. small arms and APCs
3. artillery
4. SAMs and AAA

Thus, a Red type 2 ground-to-ground weapon is also a Red type 2 weapon, and is also a Red type 2 resource. The list of Blue weapons is:

1. small arms and APCs
2. artillery
3. tanks

The list of Red personnel is:

1. tank crewmen
2. other personnel

The list of Red support resources is:

1. ammunition
2. fuel and other consumables
3. tank crewmen
4. other personnel

Thus, Red type 2 personnel are also Red type 4 support resources and Red type 9 resources. Notice that the category of Blue ground-to-air weapons is empty. (Hence, the list of Blue weapons is identical to the list of Blue ground-to-ground weapons.) Any category except ground-to-ground weapons may be empty. It is permissible for a side to have only one type of ground-to-ground weapon, which would probably be not a physical entity but an abstract measure of strength.

A unit's type determines what types of resources it can possess. The game design data fix *nrst*(i), the number of different types of resources a unit of type i can have, and *iars*(*,i), a list of the types of resources it can have. Continuing the example above, suppose:

$$nrst(5) = 6$$

$$\begin{aligned} iars(1,5) &= 7 \\ iars(2,5) &= 6 \\ iars(3,5) &= 8 \\ iars(4,5) &= 9 \\ iars(5,5) &= 5 \\ iars(6,5) &= 2 \end{aligned}$$

This says that a unit of type 5 can have 6 types of resources: Red type 7 resources (fuel and other consumables), Red type 6 resources (ammunition), Red type 8 resources, Red type 9

resources, Red type 5 resources (trucks), and Red type 2 resources (small arms and APCs). The order in which the resource types appear in *iars*(*,5) affects only the order in which IDAHEX lists the resources of type 5 units internally. By keeping the elements of *nrst* small in value, the game designer can achieve substantial economies in computer storage utilization.

The value of *iars*(*,5) in the preceding example is reasonable if a type 5 unit is a Red transport unit (as in the example at the start of Section 2.2). Notice that *iars*(*,5) lists some resources for which a transport unit would have no use, such as tank crewmen. IDAHEX allows transfers of resources between units, and therefore resources might be attached to a unit simply to move them from one place to another. If *iars*(*,5) excluded tank crewmen, a type 5 unit could not accept them and therefore could never be used to take tank crewmen to a unit that needed them.

If i is the identification number of some battle unit, the design datum [*resources*](i,j) is defined as the quantity of type j resources in the unit at the start of the game. Of course, this quantity must be 0 if the unit is prohibited from having type j resources--i.e., if there is no $1 \leq k \leq$ *nrst*(*butype*(i)) such that *iars*(k,*butype*(i)) = j. At any time during the game, [resources](i,j) is the quantity of type j resources in unit i; it equals [*resources*](i,j) at the start of the game.

The design datum *toe*(k,j) is defined as the planned effective quantity of type j resources in a type k battle unit. It might be based on the Table of Organization and Equipment for a type k unit. IDAHEX compares a unit's actual quantities of resources (given by [resources]) with its planned effective quantities in allocating supplies and replacements to it, estimating its strength, and estimating the size of its area of influence. Of course, *toe*(k,j) should be 0 if a type k unit is prohibited from having type j resources.

2.3 TIME

At the start of a game, the current time, t, equals *tinit*, which should be a nonnegative number. The game ends when t = *tend* or when a player stops it.

Time is divided into equal-length intervals called "cycles", which are subdivided into equal-length "periods", which are subdivided into equal-length "frames". Cycles, periods, and frames may all be the same length, but generally frames are shorter than periods. A "break" occurs at the start of each cycle, the start of each period, and the end of each frame. Each break causes execution of a procedure selected according to the cause of the break: at the start of each cycle, IDAHEX

inspects the game design data for changes in *mapter*, *maprd*, *maprr*, and *mapbar*; at the start of each period, it accepts players' commands; at the end of each frame, it assesses engagements, repairs, supplies consumption, and special activities.

An "event" is a break or a change in a unit's status. At t = *tinit* (the start of the game), IDAHEX ascertains when the first event will occur. It advances t to that time (possibly the same as the current time) and lets the event occur. It then ascertains when the next event will occur, advances t to that time, and lets the event occur. It continues to advance t in jumps until t ≥ *tend* or a player stops the game after a break.

In some cases IDAHEX reports the game time to the player or the game designer simply by writing the value of t; in other cases, it re-interprets t as a date and clock time, and writes them. The re-interpretation is performed by the subprogram clock, whose actual parameters are two character string variables of length 4 and the variable t. The present version of clock assumes that t measures time in days. It assigns the first character string variable a string showing the number of full days that have elapsed since the start of the war; it assigns the second a string giving the time of day in hours and minutes. If in fact, t does not measure time in days, the subprogram clock should be rewritten.

3. CHANGES IN BATTLE UNIT STATUS

A "task force" is a collection of one or more battle units --"task force elements"--that have the same status (posture, location, and objective) and will continue to have the same status as long as they remain in the task force. The elements of a task force must all belong to the same side. Each task force is identified by a positive integer; it is impossible to tell from this number alone the side to which the task force belongs.

3.1 EVENT SEQUENCING

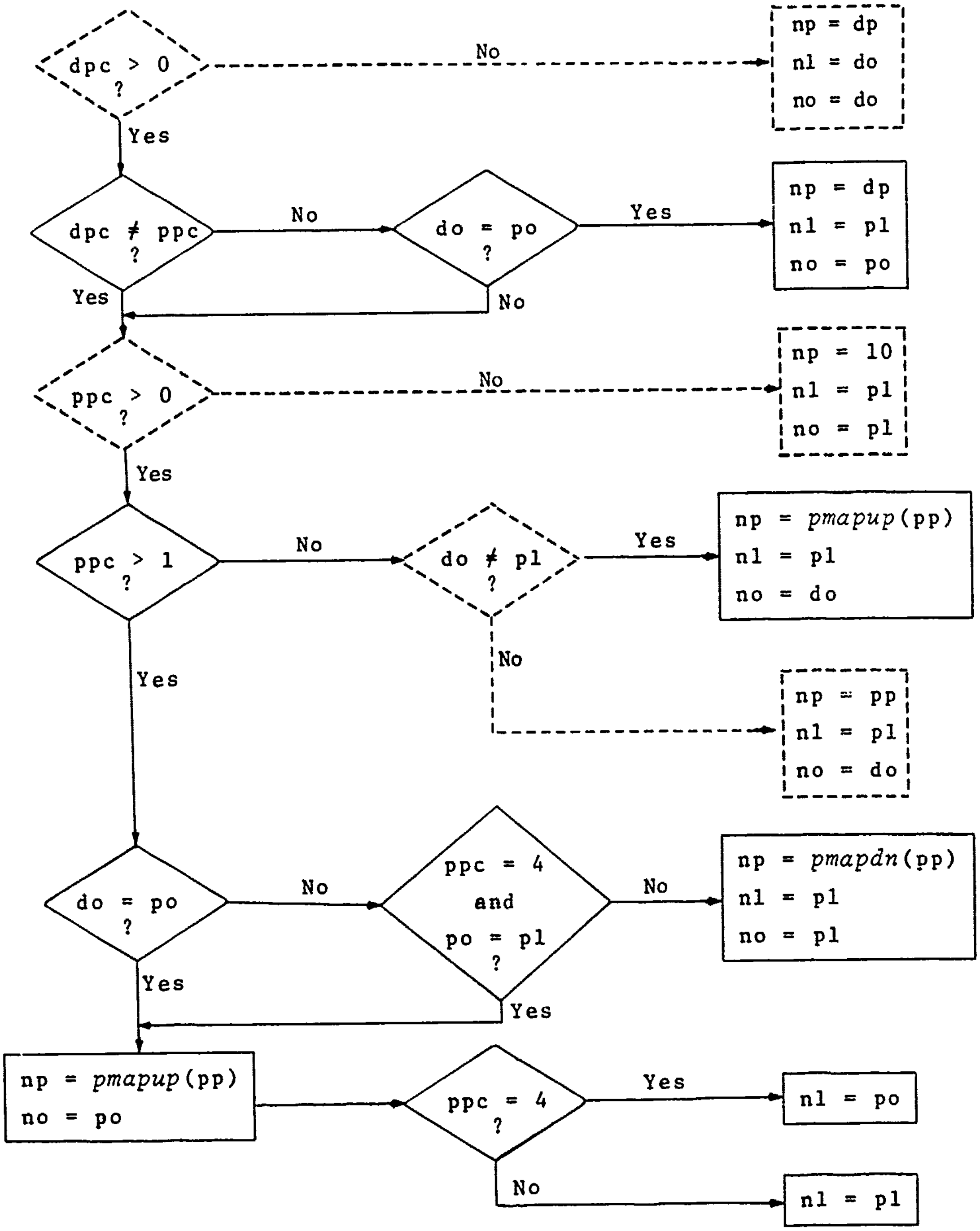
A task force's change of status is always caused and directed by an "order". Sometimes orders are generated by IDAHEX; usually they are input by the players. An order has two components: the desired objective and the desired posture. Associated with an order may be a "start time", the earliest time at which the task force should begin executing the order. Execution of an order is a process that may span time and may involve a sequence of status changes. The time required o go from one status to the next may be 0, but the task force still enters each status in the sequence. Given a task force's current status and its "active order"--the order it is executing--the logic of Figure 3.1 determines its next status. (Also see Figure 3.2.)

In some cases the task force's next status depends upon *pmapup* or *pmapdn*; specifically, its next posture is *pmapup*(pp) or *pmapdn*(pp), where pp is its present posture. IDAHEX initializes these variables as follows:

$$pmapup(\text{pp}) = \begin{cases} 20; & 10 \le \text{pp} \le 19 \\ 30; & 20 \le \text{pp} \le 29 \\ 40; & 30 \le \text{pp} \le 39 \\ 10; & 40 \le \text{pp} \le 49 \end{cases}$$

$$pmapdn(\text{pp}) = \begin{cases} -10; & 10 \le \text{pp} \le 19 \\ 40; & 20 \le \text{pp} \le 49 \end{cases}$$

The game designer can modify these values, but the modified values must be such that:

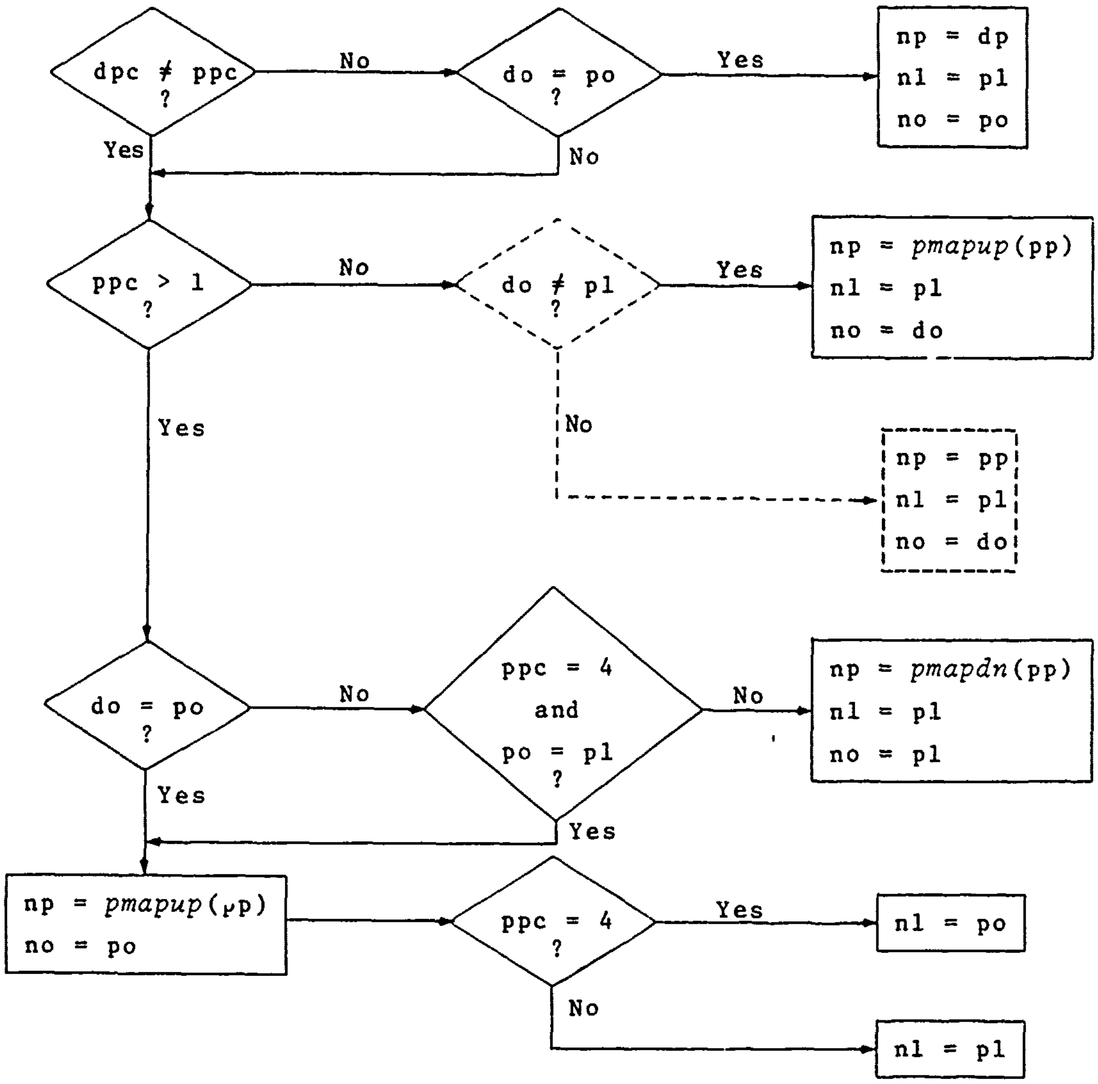


pp = present posture
ppc = present posture class
pl = present location
po = present objective

dp = desired posture
dpc = desired posture class
do = desired objective

np = next posture
nl = next location
no = next objective

Figure 3.1. STATUS SEQUENCING



pp = present posture
ppc = present posture class
pl = present location
po = present objective

dp = desired posture
dpc = desired posture class
do = desired objective

np = next posture
nl = next location
no = next objective

Figure 3.2. STATUS SEQUENCING FOR TASK FORCE WHOSE PRESENT POSTURE CLASS > 0 AND DESIRED POSTURE CLASS > 0

$$20 \le pmapup(\mathrm{pp}) \le 29 \quad \text{if} \quad 10 \le \mathrm{pp} \le 19$$
$$30 \le pmapup(\mathrm{pp}) \le 39 \quad \text{if} \quad 20 \le \mathrm{pp} \le 29$$
$$40 \le pmapup(\mathrm{pp}) \le 49 \quad \text{if} \quad 30 \le \mathrm{pp} \le 39$$
$$10 \le pmapup(\mathrm{pp}) \le 10 \quad \text{if} \quad 40 \le \mathrm{pp} \le 49$$

$$-10 = pmapdn(\mathrm{pp}) \quad \text{if} \quad 10 \le \mathrm{pp} \le 19$$
$$40 \le pmapdn(\mathrm{pp}) \le 49 \quad \text{if} \quad 20 \le \mathrm{pp} \le 49$$

The positive posture classes form a cyclic set, and *pmapup*(pp) is the posture a task force enters when it transitions to the next higher posture class: from posture class 1 it goes to 2, from 2 to 3, from 3 to 4, and from 4 to 1. The variable *pmapdn* is not used to take a task force from its present posture to the next lower posture class--that is generally illegal. Rather, it tells what posture a disengaging, moving, or attacking task force enters when it aborts the disengagement, movement, or attack and attempts to revert to a hold posture at its present location. Table 3.1 contains examples of status sequencing based on the area of war depicted in Figure 3.3. To get more specific examples, let

$$npost(1) = 4,\ npost(2) = 1,\ npost(3) = 2,\ npost(4) = 3,$$

and set *pmapup* and *pmapdn* as follows:

pp	*pmapup*(pp)	*pmapdn*(pp)
10	20	-10
11	20	-10
12	21	-10
13	21	-10
14	21	-10
20	30	42
21	31	42
30	40	42
31	41	42
40	11	42
41	12	42
42	14	42

The preceding assignments are motivated by the following interpretations of the postures:

10 standard defense
11 halted, dispersed off-road
12 halted, mainly on roads
13 prepared for transferring resources
 to other units (*itrfp* = 13)
14 hasty, disorganized defense
20 standard disengagement
21 disengagement mainly by road

Table 3.1. EXAMPLES OF STATUS CHANGES
(Refer to Figure 3.3)

task force elements	present location	present posture	present objective	desired posture	desired objective	next location	next posture	next posture class	next objective
4,9	17	10	17	10	13	17	*pmapup*(10)	2	13
4,9	17	20	13	10	13	17	*pmapup*(20)	3	13
4,9	17	30	13	10	13	17	*pmapup*(30)	4	13
4,9	17	40	13	10	13	13	*pmapup*(40)	1	13
4,9	17	12	17	10	13	17	*pmapup*(12)	2	13
4,9	17	11	17	15	17	17	15	1	17
4,9	17	10	17	22	13	17	*pmapup*(10)	2	13
4,9	17	15	17	22	13	17	*pmapup*(15)	2	13
4,9	17	32	16	41	16	17	*pmapup*(32)	4	16
4,9	17	32	16	10	16	17	*pmapup*(32)	4	16
21	6	12	6	10	6	6	10	1	6
21	6	12	6	31	9	6	*pmapup*(12)	2	9
21	6	31	9	40	9	6	*pmapup*(31)	4	9
21	6	43	9	40	9	6	40	4	9
21	6	40	9	10	9	9	*pmapup*(40)	1	9
21	6	40	9	10	6	6	*pmapdn*(40)	4	6
21	6	42	9	11	6	6	*pmapdn*(42)	4	6
21	6	31	9	10	6	6	*pmapdn*(31)	4	6
21	6	23	9	14	6	6	*pmapdn*(23)	4	6
21	6	44	6	10	6	6	*pmapup*(44)	1	6
21	6	44	6	11	6	6	*pmapup*(44)	1	6

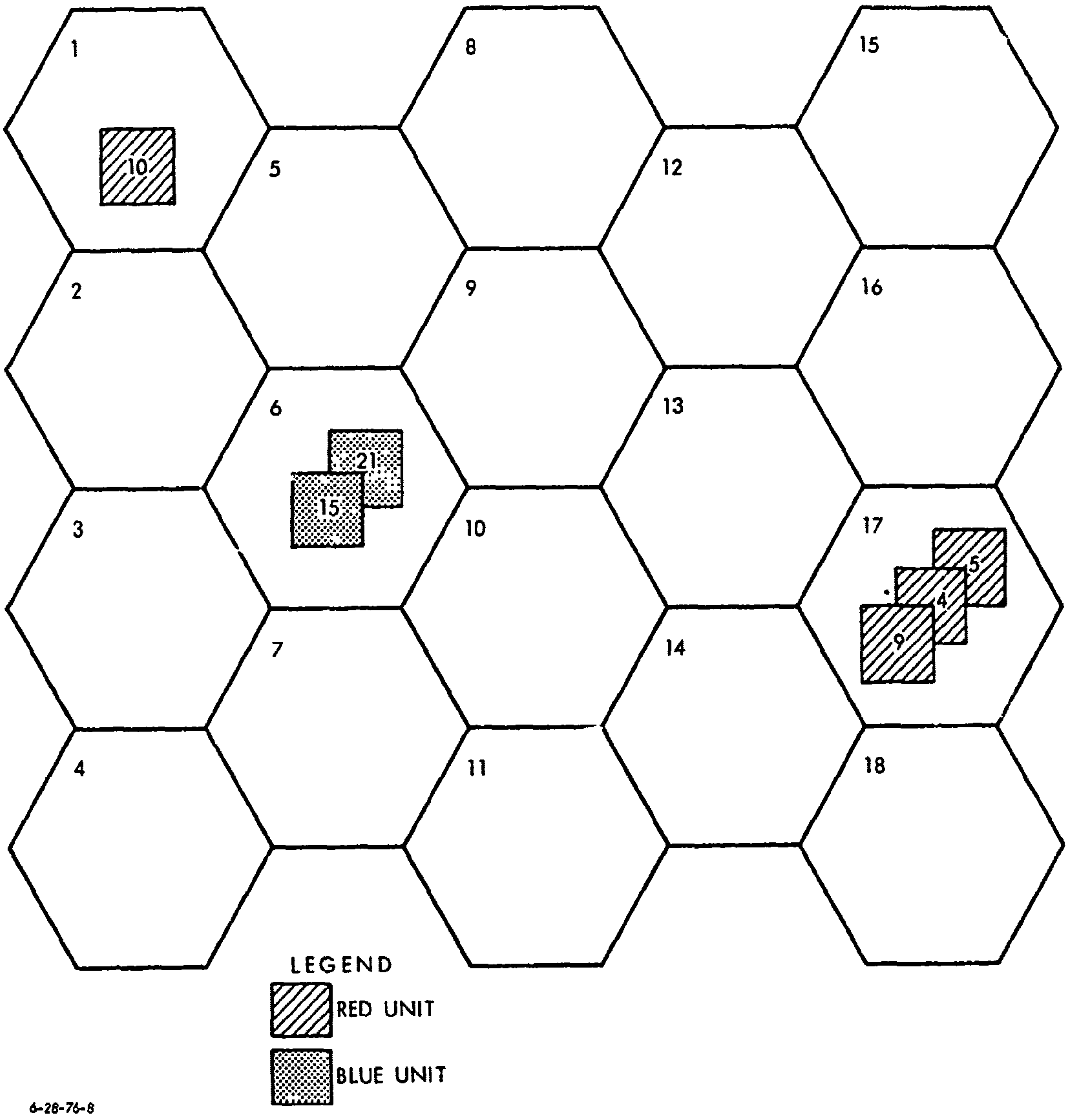


Figure 3.3. AREA OF WAR WITH BATTLE UNITS

30 tactical march
31 administrative march
40 standard attack
41 attack from administrative march
42 hasty, disorganized attack

Presumably, the ground combat attrition data make a unit less effective on defense in posture 12 than posture 11, and less effective in posture 11 than posture 10. Likewise, an attacker should be less effective in posture 41 than posture 40. Based on the above values of *pmapup* and *pmapdn* and the area of war in Figure 3.3, Table 3.2 shows the sequence of statuses induced by various orders. The last example in the table depicts a task force aborting an attack and reverting to a hold posture at its present location.

The preceding configuration can be simplified: let *npost*(1) = 2, *npost*(2) = *npost*(3) = *npost*(4) = 1, and accept the default values of *pmapup* and *pmapdn*. Let *itrfp* = 11. Then a task force in a hold posture in cell 6 whose desired posture is 11 and desired objective is 9 would go through the following sequence of statuses:

location	posture	objective
6	20	9
6	30	9
6	40	9
9	10	9
9	11	9

When the task force achieves posture 11, it will be ready and able to transfer resources to friendly units located in cell 9. If the movement of supplies and replacements is to be played explicitly, one hold posture should normally be set aside as a transfer posture, identified by the number *itrfp*. No transfer posture exists if the game designer selects *itrfp* so that *npost*(1) < *itrfp* ≤ 19. A task force whose

location = 6,
posture = 40,
objective = 9,

and whose

desired posture = 10,
desired objective = 6,

would go through the following sequence:

location	posture	objective
6	40	6
6	10	6

Table 3.2. EXAMPLES OF STATUS SEQUENCES

present location	present posture	present objective	desired posture	desired objective	Sequence of Statuses		
					location	posture	objective
6	10	6	10	9	6	20	9
					6	30	9
					6	40	9
					9	11	9
					9	10	9
6	11	6	31	9	6	20	9
					6	30	9
					6	31	9
6	31	9	10	9	6	41	9
					9	12	9
					9	10	9
6	31	9	40	9	6	41	9
					6	40	9
6	0	5	10	6	6	10	6
6	40	9	10	6	6	42	6
					6	14	6
					6	10	6

Thus, the task force aborts an attack and goes directly into the standard hold posture at its location; in contrast to the last example in Table 3.2, there is no "disorganized defense" posture in which to put it. Because this difficulty can arise whenever the game designer selects a skeleton configuration of postures, IDAHEX provides another way of reducing a task force's defensive capability in this situation: the task force can be credited with negative defense preparation time.[1]

In every example of task force movement thus far, the objective has been a cell adjacent to the task force's location, but Figures 3.1 and 3.2 do not require that. A task force may receive an order stating a desired objective not adjacent to its location. The task force will be able to execute the order only if it is airmobile. Air movement is discussed in the next discussion.

3.2 EVENT SCHEDULING

Associated with any change of status is a delay time. The task force undergoing the change stays in its present status a length of time equal to the delay, and then enters its next status. The delay is computed by the IDAHEX function wait, *which is designed for easy modification or replacement.*

Throughout this subsection, $u_1,...,u_n$ are the unit numbers of the task force elements. The side to which they belong is s; s = 1 if they are Red, s = 2 if they are Blue. The task force's location is cell pl. Its posture is pp. Its posture class is ppc. (ppc = [floor](pp/10).) Its objective is cell po. Its next location is nl, its next posture is np, its next posture class is npc (npc = [floor](np/10)), and its next objective is no. The preceding subsection reveals how the task force's present status (location pl, posture pp, objective no) and its active order determine its next status (location nl, posture np, objective no). This subsection reveals how the delay for the transition from the present status to the next status is determined. Let d denote that delay.

3.2.1 Transition within Positive Posture Class

npc = ppc, ppc > 0, no = po

[1] Preparation time's effect on attrition is discussed in Section 6.1.1. The way an aborted movement or attack can lead to negative preparation time is discussed in Section 3.4.6.

Let

$$j = pp - 10*ppc + 1$$

and

$$k = np - 10*ppc + 1.$$

Thus, posture pp is the j-th posture of posture class ppc, and posture np is the k-th posture. The delay is given by

$$d = \mathit{ptran}(ppc,j,k).$$

Regardless of the game design data, d = 0 if j = k.

3.2.2 From Hold Posture to Disengagement Posture

ppc = 1, npc = 2

In this case, d = 0.

3.2.3 From Disengagement Posture to Movement Posture

ppc = 2, npc = 3

The delay, d, in going from a disengagement posture to a movement posture is the "disengagement delay". It is most simply interpreted as the time required to break contact with the enemy, but in reality a force being pursued by the enemy might never break contact completely. A better interpretation of the disengagement delay is the amount by which contact with the enemy increases the time needed for the task force to relocate from cell pl to cell po. Although disengagement precedes movement in the event sequence, the calculation of the disengagement delay anticipates the posture in which the movement will occur; in some cases, the disengagement delay depends upon a prediction of the subsequent movement delay.

Step 0. If cell po is nonexistent or inactive, then set $d = +\infty$ and go to Step 5.

Step 1. If the task force is not engaged, then set d = 0 and go to Step 5. If cell po is not adjacent to cell pl, then set $d = +\infty$ and go to Step 5.

Step 2. The basic disengagement delay of a type i unit is, by definition, *diseng*(i). It reflects the unit's agility in terminating defense and breaking contact with its attackers.

Because the task force cannot have disengaged until every element has, its basic disengagement delay is computed as

$$d1 = \max \{\mathit{diseng}(\mathit{butype}(u_i)); 1 \leq i \leq n\}.$$

(Recall that the task force consists of units $u_1,\ldots,u_n$.)

Step 3. Let d2 = 0. Distinguish two situations: (1) there are friendly units in hold postures in cell pl; (2) there are no friendly units in hold postures in cell pl. (No such unit could belong to the task force since its posture would differ from the task force's.) In Situation 1 the friendly units are assumed to prevent enemy units engaged in combat with the task force from pursuing it during its movement to cell po. If Situation 1 obtains, go to Step 4.

In Situation 2, the enemy units may be able to maintain contact with the task force during its movement to the adjacent cell po. The disengagement delay is therefore the sum of d1, the delay computed in Step 2, and another term, d2, related to the predicted movement delay. The predicted movement delay, pmd, is calculated exactly as the task force's movement delay would be calculated if it were in posture np, attempting to enter an attack posture with the same objective. As in calculation of the movement delay (Section 4.1), define M to be the index set of those types of the task force's resources that move independently instead of using other resources for transportation.

The game design datum *elude*(i,s) reflects the ability of side s type i resources to elude pursuit; the lower it is, the more elusive they are; it it is 0, they are completely elusive. Since a task force behaves as an integral whole, it can only be as elusive as its least elusive independently moving resources. Let

$$ef = \max \{\mathit{elude}(i,s); i \in M\}.$$

Let

$$d2 = ef * pmd.$$

Step 4. Set d = d1 + d2.

Step 5. End

3.2.4 From Movement Posture to Attack Posture

ppc = 3, npc = 4, no = po

The delay, d, in going from a movement posture to an attack posture with the same objective is the "movement delay". It corresponds to the time the task force would need in order to move physically from its location to its objective if unimpeded by the enemy. The task force is not considered to have arrived at its location until all its resources have (equivalently, its faster resources are assumed to travel slowly enough not to outdistance other resources), but its resources may travel by different modes. The game design datum *mode*(i,s) indicates the mode--cross-country/road, air rail, or sea--in which side s type i resources travel under their own power.

Some types of resources in a task force are "independently moving"; they move under their own power rather than using other resources for transportation. The other types of resources (if any) are "passengers"; they may ride on independently moving resources. Some types of independently moving resources, called ferries, may move backward and forward along the route to the objective--carrying passengers when they move forward, then dropping them off and going back for more. The classification of the task force's resources depends on its movement posture and the types of units it contains.

The movement delay is calculated as the sum of three distinct delays:

$$d = d0 + d1 + d2.$$

The delay d0 is either 0 or $+\infty$ (in which case $d = +\infty$).[1] It enforces prohibitions against certain movements by making the time needed to accomplish them infinite. The delay d1 is proportional to the distance to be traveled, while d2 represents time spent crossing barriers.

<u>Step 0</u>. Initially, let d0 = d1 = d2 = 0. If cell po is nonexistent or inactive, let $d0 = +\infty$ and go to Step 7.

<u>Step 1</u>. Classify each type of the task force's resources as independently moving resources or passengers. Determine the types of independently moving resources that are ferries.

[1] Usually, an infinite movement delay results from a mistake by the player. If that may be the case, IDAHEX warns the player of the side to which the task force belongs, explaining why the delay is infinite.

Step 2. If cell po is not adjacent to cell pl, if it is the location of an active enemy unit, and if it is not owned by the task force's side, then let d0 = +∞ and go to Step 7.

Step 3. If the task force lacks supplies it needs in order to move, let d0 = +∞ and go to Step 7.

Step 4. If some type of passenger resources can not be carried by any of the independently moving resources, let d1 = +∞ and go to Step 7. Find cap, a number that measures the independently moving resources' aggregate capacity, and burd, a number that measures the aggregate burden the passengers impose on this capacity. If cap = 0 and burd > 0, let d1 = +∞ and go to Step 7.

Step 5. The function value speed(i,s,pp-29,pl,po) is the rate at which side s type i resources can move, under their own power, from cell pl to cell po. Its derivation is explained in Section 4.2. Note that speed(i,s,pp-29,pl,po) = 0 if side s type i resources cannot fly and cell po is not adjacent to cell pl, with the result that *the task force can move directly to a non-adjacent cell only if all its resources are airborne.* Let

$$rsmr(i) = speed(i,s,pp-29,pl,po)$$

for each $i \in M$, the index set of independently moving resources. If rsmr(i) = 0 for some $i \in M$, let d1 = +∞ and go to Step 7.

If burd ≤ cap, let

$$d1 = dist(pl,po) / \min \{rsmr(i);\ i \in M\}$$

and go to Step 6; dist(pl,po) is the straightline distance from the center of cell pl to the center of cell po. If burd > cap, the ferries must make multiple trips to deliver all the passengers to cell po, which complicates the calculation of d1. Two methods are available. Method 1 takes into account differences in the ferries' speeds, but does not permit passengers to move under their own power when not being carried. Method 2 forces the ferries all to move at the speed of the slowest ferry, but permits passengers to move under their own power when not being carried. Method 2 is never used if some ferries are moving by rail, air, or sea. Method 1 or Method 2 is used--whichever yields the smallest value of d1--if all the ferries are moving on roads or cross-country. Method 2 is relevant principally to the case of an infantry unit: personnel walk when not entrucked; to the extent they can, they carry their weapons and supplies with them at all times.

Step 6. Now d2, the delay attributable to a barrier, is calculated. (At this point, d2 = 0.) If none of the task force's independently moving resources are traveling on land, go to Step 7: no barrier can affect them. If there is no movement barrier between cells pl and po, go to Step 7.

The game design variable *barrr*(k) = .true. for $1 \leq k \leq nmb$ if and only if a type k movement barrier blocks rail traffic. If such a barrier exists between cells pl and po, and if some of the task force's independently moving resources are traveling by rail, then let $d2 = +\infty$ and go to Step 7. If no such barrier exists and all the independently moving resources are traveling by rail, go to Step 7: there is no barrier delay.

In the remaining cases, the task force's barrier delay is assessed as the maximum delay of any element with resources traveling by road or cross-country. Any such element's delay depends upon its unit type, the task force's movement posture, and the movement barrier type.

Step 7. Set d = d0 + d1 + d2. End.

3.2.5 From Attack Posture to Hold Posture at New Location

ppc = 4, npc = 1, no $\neq$ pl

The "attack delay" is 0 if cell po contains no enemy units in hold postures. Otherwise, the delay is indefinite: it depends upon the course of combat.

3.2.6 Reorientation to Present Location

ppc > 1, npc > 0, no = pl

Two cases are possible: (1) $2 \leq ppc \leq 4$, no = pl $\neq$ po; (2) ppc = 4, no = pl = po. The first case occurs when a disengaging, moving, or attacking task force tries to revert to a hold posture in its present location; its next posture is an attack posture, and its next objective is its location. And then Case (2) applies. In either case, d = 0.

3.2.7 Activation

$ppc \leq 0$, npc = 1

If ppc = -1, $d = +\infty$: a destroyed unit cannot come base to life. If ppc = 0, d = 0.

3.2.8 Transition to or within Nonpositive Posture Class

$np < 10$

The delay is 0. Since there is normally no reason for a unit to enter posture class 0 from another posture class, a warning is issued to the game designer if that happens. The warning message is placed in the game designer's output file, file 51.

3.3 EVENT RESCHEDULING

A task force's movement delay may depend upon such conditions as the environments in its location and its objective, the type of road link (if any) between the cells, the type of barrier (if any) between them, and the supplies it has. The movement delay calculation takes these conditions as they are when the task force begins moving (to be precise, when it begins the transition from a movement posture to an attack posture). But these conditions may change during a movement, making re-evaluation of the movement delay desirable.

Supplies consumption is assessed at the end of every frame. If at that time a moving task force is found to have no supplies of some type, its movement delay is re-evaluated as though it were just beginning the movement, in its present posture. If the re-evaluated delay equals or exceeds 10**9, the movement is aborted: the task force is given the active order specifying 10 as the desired posture and its present location as the desired objective.

The variables *mapter, maprd, maprr,* and *mapmb* may change at the beginning of every cycle, possibly changing the environment, road, rail and barrier types on which a movement delay was based. This is not a significant difficulty provided movement delays are short relative to the cycle length, as would be the case in most games. But to accommodate the remaining cases, IDAHEX re-evaluates movement after any change in *mapter, maprd, maprr,* or *mapmb*. Incidentally, the re-evaluation detects the effects of LOC modification activities (damage, repair, or improvement of lines of communication).

To re-evaluate movement delays, IDAHEX must keep track of the fraction of each task force's movement that has been completed. If task force m is moving (awaiting transition from a movement posture to an attack posture with the same objective), mncomp(m) is defined to be the fraction of its movement (from its present location to its present objective) it has already completed; mncomp(m) was set to 0 when the task force began moving. When *mapter, maprd, maprr,* or *mapmb* changes, the

movement delay is re-evaluated as follows:

Step 0. Let tlast be the last time at which the task force's movement delay was evaluated (or re-evaluated). If the task force's active order has a start time that exceeds tlast, redefine tlast to be the start time. Let tfin be the time at which the task force's next change of status (transition to an attack posture) is scheduled to occur. (Recall that t is the current time.) Estimate the fraction of the movement completed since time tlast as

$$\text{delta} = \begin{cases} (t-\text{tlast})/(\text{tfin}-\text{tlast}); & \text{tlast} < t \\ 0 & ; \text{tlast} \geq t \;. \end{cases}$$

Redefine mncomp(m) accordingly:

$$\text{mncomp}(m) \leftarrow \text{mncomp}(m) + \text{delta}.$$

Step 1. Let d be the movement delay that would be calculated if the task force were just beginning its movement. Resdhedule its next change of status to occur at the time

$$\max\{t,\text{tlast}\} + (1-\text{mncomp}(m)) * d \;,$$

Although a disengagement delay may include a term proportional to an anticipated movement delay, disengagement delays are not automatically re-evaluated when *mapter, maprd, maprr,* or *mapmb* changes.

3.4 TACTICAL SITUATIONS

Because the forces can maneuver and the processes of maneuver span time, situations requiring special logic may arise. In many cases they require tactical decisions, in contrast to the players' operational decisions, and therefore should handled by IDAHEX. Handling these tactical situations with precision is not critical--indeed, that would be inconsistent with the model's level of resolution.

Section 3.1 shows how events are determined, and Section 3.2 shows how they are scheduled. The events are arranged implicitly in a queue in order of scheduled occurrence; the event scheduled to occur next is at the front of the queue. A change of status by a task force in the attack posture class comes after a change of status by a task force in another posture class if both events are scheduled for the same time. When an event comes to the front of the queue, the time, t, is advanced to the time at which it is scheduled to occur, and the event is passed to the subprogram xeq for execution. Instead of executing the event, xeq may alter the queue--

adding events to it, or changing the times at which events are scheduled to occur and changing the order of events in the queue.

Some terms are needed. To "occupy" a cell is to change location to the cell. A side's "security force" in a cell consists of every friendly unit that is located in the cell and is in a hold posture. A unit or taks force whose posture is disengagement, movement, or attack and whose objective is cell j is equivalently said to be in a disengagement, movement, or attack posture oriented toward cell j, or to be disengaging, moving, or attacking toward cell j.

The variable eps = *tframe* / 100.

3.4.1 Pursuit

Suppose a Blue task force in cell i enters a movement posture oriented toward cell j, an adjacent cell. Suppose that later a Red task force occupies cell i and subsequently enters a movement posture oriented toward cell j. If the Red task force is more mobile, its movement delay may be less than the Blue task force's delay--so much less that its movement delay ends before the Blue task force's. But because xeq implements the following rule, the Red task force cannot occupy cell j before the Blue task force.

Let task force m and task force n belong to opposite sides. Suppose the location, posture class, and objective of task force m coincide with the location, posture class, and objective of task force n. Also suppose that the next location, next posture class, and next objective of task force m coincide with the next location, next posture class, and next objective of task force n and the task forces' next posture class differs from their present posture class. Let $u_1,\ldots,u_j$ be the identification numbers of the units in task force m, and let $v_1,\ldots,v_k$ be the identification numbers of the units in task force n. If

$$\min \{\mathrm{tentry}(u_i);\ 1 \leq i \leq j\} > \min \{\mathrm{tentry}(v_i);\ 1 \leq i \leq k\},$$

then task force m may enter its next status no sooner than eps/4 after task force n.

3.4.2 Attack

An important variable in many tactical situations is [owner]; [owner](i) = 1 if cell i is owned by Red and 2 if the

cell is owned by Blue. The game design data set [*owner*], and then IDAHEX sets [owner] = [*owner*]. Thus, the design data declare the ownership of each active cell at the start of the game. This subsection shows, among other things, how [owner] gets changed.

Suppose task force m, belonging to side sa (sa = 1 or sa = 2), is in an attack posture. Let sd = 3 - sa; side sd is its enemy. Suppose the task force's location is cell pl, its objective is cell po, its next posture is np, and its next objective is no; no = pl is permitted. Assume posture np is a hold posture. Assume the task force's attack delay (possibly 0) is complete, the task force has reached the front of the queue, and the subprogram xeq has been called to execute the task force's transition to its next status, a hold posture in cell po. The rest of this subsection charts the actions taken by xeq in this case. The verb "return" means "return from xeq to the calling program".

<u>Step 1</u>. If task force m is already engaged, go to Step 6. Search for side sd task forces whose location is cell po, whose posture class is 2, 3, or 4, and whose objective is cell pl. If none exist, go to Step 2. Do the following for each such task force: make its desired objective po; make its desired posture

pmapup(post)	if it is attacking,
pmapup(*pmapup*(post))	if it is moving
pmapup(*pmapup*(*pmapup*(post)))	if it is disengaging,

where post is its posture; schedule its next change of status for time t, and place it ahead of task force m in the queue. Return. This procedure leads eventually to an engagement in which task force m is attacking side sd units holding in cell po; it obviates an entirely separate combat procedure for meeting engagements.

<u>Step 2</u>. If an engagement already exists at cell po, go to Step 7. If [owner](po) = sa, go to Step 3. Search for enemy task forces in movement or attack postures oriented toward cell po whose next change of status is scheduled to occur no later than t + eps. If none exist, go to Step 3. Reschedule the next change of status of each of these task forces to time t and place it ahead of task force m in the queue. Return. This step resolves virtual ties in times at which hostile units arrive at a cell in favor of the cell's current owner.

<u>Step 3</u>. Search for active side sd units located in cell po and not already engaged. If none exist, let task force m change status (let it occupy cell po), let [owner](po) = sa, and return. If cell po contains a side sd unit in a hold posture other than

posture *itrfp*, go to Step 4. Let S be the set of every side sd unit whose location is po and whose posture is *itrfp*. Two cases are possible. Case 1: S is nonempty. In this case, constitute every member of S that does not belong to a task force as a task force, give it the order "desired objective = po, desired posture = 10", and position it in the queue according to the time of its next change of status. Let T be the set of every task force whose elements are members of S. For each task force in T, if there is a start time associated with the task force's active order, and it exceeds t, reset it to t and therefore reschedule the task force's next change of status. Now for each task force in T, if the task force's active order specifies a hold posture in cell po and the next change of status is scheduled to occur no later than t + eps, reschedule it to occur at time t and move the task force ahead of task force m in the queue. Return. Case 2: S is empty. In this case, let T be the set of every side sd task force located in cell po whose objective is owned by side sa and whose objective contains one or more active side sa units. For each task force in T, determine whether the task force could execute the first change of status implied by the order "desired objective = po, desired posture = 10" no later than t + eps; if so, make that its active order, schedule its next change of status for time t, and place it ahead of task force m in the queue. If one or more task forces have received new orders in this way, return.

Step 4. If cell po contains no side sd security force, go to Step 5. If [owner](pl) = sd, change the active order of task force m to "desired objective = pl, desired posture = -10", schedule its next change of status for time t (keep task force m at the front of the queue), and return. (The units in task force m are destroyed because they are attacking at the same time the enemy owns their base. It is inappropriate to let their location be cell pl, and they have not been able to occupy cell po; therefore they must be removed from the area of war. Step 3 alters orders and re-sequences the queue to avert such catastrophes whenever possible.) Set up an engagement between task force m and every side sd unit in a hold or disengagement posture in cell po that is not already engaged. Reschedule the task force's next change of status to occur at time $+\infty$. Return.

Step 5. Reaching this point implies there is no side sd security force in cell po, but one or more active units from side sd are located there. Let S be the set of every side sd unit whose location is cell po and whose posture is a movement posture. For each unit $u \in S$, if tentry(u) > t - *delta*, change the unit's posture to *pmapup*(*pmapup*(*pmapup*(post))), where post is its present posture. (Side sd units that started moving from cell po within the interval *delta* before the arrival of task

force m must revert to disengagement postures.) Take each side sd unit in a disengagement posture in cell po that is not already engaged, and join it in an engagement with task force m. Take each side sd unit located in cell po that is attacking in some engagement, constitute it as a task force if not already an element of one, give the task force the active order "desired objective = po, desired posture = -10", and place the task force at the front of the queue. Let task force m enter its next status. (Let it occupy cell po.) Return.

Step 6. This point is reached if and only if task force m is already engaged. For that to happen, xeq must have been called once before to execute the task force's transition from an attack posture oriented toward cell po to a hold posture in cell po, and Step 4 must have joined the task force in an engagement. When that happened, xeq did not let the task force enter its next status; in fact, it rescheduled the change of status to occur after the end of the game. Subsequently, the change of status was rescheduled as Section 3.4.3 explains, and task force m again reached the front of the queue, inducing the current invokation of xeq. Proceed as follows. Take each side sd unit located in cell po that is in an attack posture and engaged, constitute it as a task force if it does not already belong to one, give the task force to which it belongs the active order "desired objective = po, desired posture = -10", and place the task force at the front of the queue. Let task force m enter its next status. (Let it occupy cell po.) Return.

Step 7. (This step is reached only if there is already an engagement at cell po.) If there is no engagement at cell po in which side sa is the attacker, go to Step 7(b).

Step 7(a). Join task force m to the engagement at cell po in which its side is the attacker. Reschedule its change of status to occur at time $+\infty$. Return.

Step 7(b). If side sd has a security force in cell po, remove it from the engagement already in progress there. Otherwise, remove every (disengaging) side sd unit from the engagement. If no side sd units remain in the engagement, then terminate the engagement, and re-evaluate the status change delay of every disengaging task force whose elements were participating in the engagement (including the side sd units just removed from it). If any of the re-evaluated delays are 0, then place every task force whose re-evaluated delay is 0 ahead of task force m in the queue, and return. Go to Step 3.

3.4.3 Disappearance of a Security Force

Suppose cell pl is owned by side s (s = 1 or s = 2) and contains one or more units from side s in hold postures, and

suppose one or more units from side 3-s are attacking cell pl. Suppose a task force consisting of the entire side s security force in cell pl now enters posture class -1, 0, or 2. Then the delay of every side s task force located in cell pl and disengaging is re-evaluated, possibly casuing rescheduling of the task force's next change of status. This is necessary because a delay computed when a friendly security force existed might no longer be appropriate; in particular a disengagement delay might have to be extended now that the enemy can pursue. Next, the active order of every side 3-s task force attacking toward cell pl is inspected. If the order implies that the task force's next change of status is something other than just a transition to another attack posture oriented toward cell pl, the change of status is rescheduled to t and moved to the front of the queue (giving the task force the opportunity to occupy cell pl). Otherwise, the order is discarded, so that the next order, if any, in the task force's mission becomes the active order, and the test is repeated.[1] The process continues until either the test is passed or no orders remain in the task force's mission.

3.4.4 Counterattack

IDAHEX structures every engagement in such a way that units from one side are attacking and units from the other side are defending. A defender is in a hold posture or a disengagement posture. It is possible that all defenders in the engagement are holding, or all disengaging, or some holding and some disengaging. The defenders are all located in the same cell, the cell under attack, while the attackers may be located in different cells. In a counterattack, a task force consisting of one or more defenders disengages, moves, and attacks toward the location of one or more of the attackers. Suppose the defenders' location is cell i. Suppose task force m consists of one or more of the defenders in hold postures, and xeq has been called to execute its transition to a disengagement posture oriented toward cell j, the location of one or more of the attackers. Let A be the set of the attackers located in cell j. If task force m is not stronger than A, the task force's active order is changed to "desired objective = i, desired posture = 10", its next change of status is scheduled for time t, and it is placed at the front of the queue. If task force m is stronger than A, A's attack is aborted: each unit in A that does not belong to a task force is constituted as one; each task force contained in A is given the active order

[1] Missions are explained in Section 4. Basically, a mission is a sequence of orders for a task force.

"desired objective = j, desired posture = 10", its next change of status is scheduled for time t, and it is placed ahead of task force m in the queue; xeq returns without executing the transition of task force m to its next status. The criterion for deciding whether task force m is stronger than A is as follows. Let $u_1,\ldots,u_n$ be the task force's elements, identified by unit numbers. Let s = 1 if the task force is Red and s = 2 if it is Blue. The attack strength of task force m is given by

$$f0 = \sum_{k=1}^{n} \sum_{irs=1}^{nrs(s)} rsvala(irs,s) * [resources](u_k,irs).$$

The number rsvala(irs,s) is the standard value of a side s type irs resource on attack; its computation is explained in Section 6.3. Basically, rsvala(irs,s) measures the contribution of a single type irs resource belonging to a standard side s force attacking a standard enemy force in a standard engagement. The defense strength of task force m is given by

$$g0 = \sum_{k=1}^{n} \sum_{irs=1}^{nrs(s)} rsvald(irs,s) * [resources](u_k,irs).$$

The number rsvald(irs,s) is the standard value of a side s type irs resource on defense. Let $v_1,\ldots,v_r$ be the units in the set A, identified by their numbers. Let s′ = 3 − s. The attack strength of A is given by

$$f1 = \sum_{k=1}^{r} rsvala(irs,s') * [resources](v_k,irs).$$

The defense strength of A is given by

$$g1 = \sum_{k=1}^{r} rsvald(irs,s') * [resources](v_k,irs).$$

Task force m is considered stronger than A if and only if

$$\frac{f0}{g1} > \frac{f1}{g0}.$$

3.4.5 Activation of Inactive Task Force

Suppose xeq has been called to execute the transition of a task force whose elements are in a nonpositive posture class

to a positive posture class. If the task force's next location (the cell where it will become active) is owned by the enemy, or if one or more active enemy units are located there, xeq does not execute the change of status and, instead, reschedules it for $t = +\infty$ and warns the player of the side to which the task force belongs.

3.4.6 Virtual Time of Posture Class Entry

When a task force transitions from its present status to its next status, tentry may be reset for each of its elements. Let pp be the task force's posture, ppc its posture class, po its objective, and pl its location. Let npc be its next posture class and no is next objective. Let units $\{u_i;\ 1 \leq i \leq n\}$ be its elements.

If npc = ppc, tentry is not changed. Henceforth, assume npc $\neq$ ppc.

If npc = 2 or npc = 3, IDAHEX sets

$$\text{tentry}(u_i) = t$$

for every $1 \leq i \leq n$. That is, the virtual time at which the units enter their next posture class equals the actual time.

If npc = 1, or if npc = 4 and no $\neq$ pl, IDAHEX sets

$$\text{tentry}(u_i) = \max\ \{t,\ \text{tentry}(u_i)\}$$

for every $1 \leq i \leq n$.

In the remaining case, npc = 4 and no = pl; the task force is aborting a disengagement, movement, or attack and trying to revert to a hold posture in cell pl. If there is no enemy task force whose objective is pl, whose location is not po, and whose posture class is 2, 3, or 4, then

$$\text{tentry}(u_i) \leftarrow \max\ \{t,\ \text{tentry}(u_i)\}$$

for every $1 \leq i \leq n$. Otherwise, tentry is determined as follows. If ppc = 2, then

$$\text{tentry}(u_i) \leftarrow \max\ \{t,\ \text{tentry}(u_i)\}$$

for every $1 \leq i \leq n$. If ppc = 3, then

$$\text{tentry}(u_i) \leftarrow \max\ \{t + (t-\text{tentry}(u_i)),\ t\}$$

for every $1 \leq i \leq n$; that is, the virtual time of entry is set ahead of the current time by the length of time the unit has been in a movement posture. Finally, if pcc = 4, tentry (u_i) is, for every $1 \leq i \leq n$, set equal to t plus the movement delay that would be computed for the (entire) task force were it moving from cell po to cell pl in posture (*pmapup*(*pmapup*(pp))).

Thus, if a task force aborts a movement or attack and an enemy unit directly threatens to seize its location from the flank or rear, tentry for its elements is set ahead in time to indicate just how far out of position it is. Because of the way tentry is set for transitions into posture class 1, this penalty is retained when the task force subsequently reverts to holding its present location. The combat procedure uses t - tentry(j) as a measure of the length of time unit j has had to prepare a defense; if unit j has aborted a movement or attack and reverted to holding its location, its preparation time may be negative.

3.4.7 Engagment Termination

Suppose engaged task force m goes from a disengagement posture to a movement posture, or enters a nonpositive posture class (its units are destroyed or de-activated), or breaks off an attack and tries to revert to a hold posture in its own location. Then xeq deletes the task force's elements from their engagment. If no units from their side remain in the engagement, then the engagement terminates. In this event, xeq may reschedule the times at which enemy units that were engaged enter new statuses. Suppose task force n, an enemy of task force m, was participating in the terminated engagement. The time at which it is scheduled to enter its next status is reset to min {t0,t1}, where t0 is the time at which it is presently scheduled to enter its next status, and t1 is the time at which the task force (which is now not engaged) would enter its next status if it were just beginning its transition to its next status. The result is that disengaging task forces can immediately enter movement postures.

4. MOVEMENT

This section explicates the derivation of a task force's movement delay, d. Let $u_1,\ldots,u_n$ be the unit numbers of the task force elements. Let s = 1 if they are Red and s = 2 if they are Blue. The task force's location is cell pl, its objective is cell po, its posture is pp ($30 \le pp \le 39$), and its next posture is np ($40 \le np \le 49$).

The task force is not considered to have arrived at its location until all its resources have (equivalently, its faster resources are assumed to travel slowly enough not to outdistance other resources), but its resources may travel by different modes. In fact, the game design variable *mode* fixes the resources' modes of travel. The game designer should define *mode* so that *mode*(i,s) = 2 if and only if side s type i resources are aircraft, 3 if and only if they are railraod rolling stock, 4 if and only if they are watercraft, and 1 otherwise. Having *mode*(i,s) = 1 indicates that side s type i resources propel themselves on roads or cross-country, or that they have no intrinsic movement capability; such resources include, for example, trucks, tanks, APCs, personnel and supplies.

As Section 3.2.4 explains, a task force's resources of a given type are either independently moving resources or passengers, and independently moving resources may or may not be ferries. The game design variables that determine the classification are *ldclas, fercl,* and *imclas* [1] The variable *ldclas* gives each type of resources the attribute "load class". The variables *imclas* and *fercl* determine the independently moving resources and the ferries by load class.

The movement delay is calculated as the sum of three distinct delays:

$$d = d0 + d1 + d2.$$

The delay d0 is either 0 or $+\infty$ (in which case $d = +\infty$). The delay d1 is proportional to the distance to be traveled, while d2 represents time spent crossing barriers.

[1] Section 4.3 illustrates how these variables might be defined.

Section 4.1 explains the procedure for finding d. Section 4.2 explains a subprocedure for finding the speeds of the individual resources.

Recall that a movement posture is numbered between 30 and 39, inclusive. Let

$$kmp = pp - 29.$$

4.1 CALCULATING THE MOVEMENT DELAY

Step 0. Initially, let d0 = d1 = d2 = 0. If cell po is nonexistent (po < 1 or po > *ncells*) or inactive, then let d0 = +∞ and go to Step 7.

Step 1. Let s = 1 if the task force is Red, and s = 2 if it is Blue.

Let

$$qrs(i) = \sum_{k=1}^{n} [resources](u_k, i)$$

for each $1 \le i \le nrs(s)$. Let

$$R = \{i: \ 1 \le i \le nrs(s), \ qrs(i) > 0\},$$

the set of resource types represented in the task force.

Let

$$lc = \mathit{fercl}(kmp).$$

Determine the index set of ferries, denoted K, as

$$K = \{i \in M: \ \mathit{ldclas}(i,s) = lc, \ \mathit{ldcap}(i,lc,s) > 0\}.$$

Let

$$m = \min \{\mathit{imclas}(kmp, \mathit{butype}(u_i)); \ 1 \le i \le n\}.$$

Determine the index set of independently moving resource types, denoted M, as

$$M = \{i \in R: \ \mathit{ldclas}(i,s) \ge m\} \cup K.$$

Let

$$L = \{i \in M: \ i \notin K\}.$$

Finally, let

$$P = \{i \in R: \ i \notin M\}.$$

P is the index set of "passengers"--resources that use other resources in the task force for locomotion. P may be empty.

Step 2. Suppose: (1) cell po is not adjacent to cell pl; (2) cell po is the location of an active enemy unit; and (3) cell po is not owned by side s--i.e., [owner](po) $\neq$ s. Then let d0 = $+\infty$ and go to Step 7.

Step 3. This step consists of ascertaining whether the task force has the supplies it needs in order to move; if *nss*(s) = 0, go to Step 4. For every $1 \le k \le$ *nss*(s), let

$$\text{demand}(k) = \sum_{i \in M} \mathit{ssreqm}(k,i,s) * \text{qrs}(i).$$

If demand(k) > qrs(nequip(s)+k) for some $1 \le k \le$ *nss*(s), let d0 = $+\infty$ and go to Step 7.

The preceding test is crude since demand(k)--the amount of type k supplies required for movement--is independent of the time needed to complete the movement (which is not yet ascertained). Perhaps the best strategy is to make *ssreqm* underestimate the supplies needed for movement. One risks letting a task force change location when it has insufficient supplies to complete the movement, but if *tframe* is suitably small, the risk is minor. At the end of each frame, supplies consumption is assessed, and if a moving task force exhausts any type of supplies, its movement delay is re-evaluated. If the delay is found to be $+\infty$, the movement is aborted, and the task force tries to revert to a hold posture in its present location.

Step 4. The aggregate load that the passengers place upon the task force's carrying capacity is, by definition,

$$\text{burd} = \sum_{i \in P} \mathit{ldreq}(i,lc,s) * \text{qrs}(i).$$

(A sum with no terms is defined to be 0; hence, burd = 0, if the set P is empty.) If *ldreq*(i,lc,s) > 10**5 for some i $\in$ P, infer that side s type i resources cannot be carried in load class lc. in this case, set d1 = $+\infty$ and go to Step 7.

The extent of the passenger load that can be accomodated by the non-ferry independently moving resources is

$$\text{capl} = \sum_{i \in L} \mathit{ldcap}(i,lc,s) * \text{qrs}(i).$$

Redefine burd as the load that remains to be carried by the ferries:

$$\text{burd} \leftarrow \text{burd} - \text{capl}.$$

The ferries' carrying capacity with respect to load class lc is

$$\text{cap} = \sum_{i \in K} ldcap(i,lc,s) * qrs(i).$$

If cap = 0 and burd > 0, set dl = +∞ and go to Step 7.

<u>Step 5</u>. The function speed gives the rate at which a given type of resources can move, under its own power, over a given route; speed(i,s,kmp,pl,po) is the movement rate of side s type i resources going from cell pl to cell po in movement posture kmp. The function is explained in Section 4.2.

Let rsmr(i) = speed(i,s,kmp,pl,po) for each i ε M. If rsmr(i) = 0 for some i ε M, let dl = +∞ and go to Step 7.

<u>Step 5(a)</u>. If burd ≤ cap, let

$$dl = \text{dist}(pl,po) \;/\; \min\{\text{rsmr}(i);\ i \in M\}$$

and go to Step 6; dist(pl,po) is the straightline distance from the center of cell pl to the center of cell po (which equals *depth* if the cells are adjacent). If burd > cap, the ferries must make multiple trips to deliver all the passengers to cell po, which complicates the calculation of dl. For road/cross-country movement, either of two methods--Step 5(b) or Step 5(c) --is used, whichever yields the smallest value of dl. For rail, air, and sea movement, only the method of Step 5(b) is used. The method of Step 5(b), taking into account the differences in the ferries' speeds, allocates a passenger burden to each type of ferry so as to minimize the time at which all ferries and all passengers have arrived at the destination. It assumes that passengers do not move under their own power. The method of Step 5(c) assumes that the ferries all move at the same rate, the speed of the slowest ferry, so that their carrying capacity is essentially homogeneous. It segregates the passengers into two classes--those who can move under their own power and those who can not--and allocates carrying capacity to each class of passengers so as to minimize the time at which all ferries and all passengers have arrived at the destination. If rsmr(i) is indeed the same for every i ε K, Step 5(c) yields at least as early an arrival time as Step 5(b).[1]

---

[1]Two alternative methods are used (continued on next page)

Let dl = +∞ and dc = +∞.

Step 5(b). Choose il ε K. Let

$$pl = \frac{burd - templ/2 + temp2/(2*rsmr(il))}{1 + temp2/temp3},$$

where

$$templ = \sum_{\substack{i \varepsilon K, \\ i \neq il}} qrs(i) * ldcap(i,lc,s),$$

$$temp2 = \sum_{\substack{i \varepsilon K, \\ i \neq il}} rsmr(i) * qrs(i) * ldcap(i,lc,s),$$

$$temp3 = rsmr(il) * qrs(il) * ldcap(il,lc,s).$$

Next, set

$$dl = (dist(pl,po) / rsmr(il)) * (2/rho - 1),$$

where

$$rho = qrs(il) * ldcap(il,lc,s) / pl.$$

Theorem 1 of Section 4.4. justifies this formula and clarifies the assumptions behind it; dist(pl,po) corresponds to the Proposition's D, qrs(i) corresponds to b_i, *ldcap*(i,lc,s) corresponds to c_i, and rsmr(i) corresponds to s_i.

Step 5(d). Compute dl as the lesser of the delay calculated by the method of Step 5(b) and the method of Step 5(c):

$$dl \leftarrow \min \{dc, dl\}.$$

Now dl is the time needed for all the ferries and all the passengers to reach the objective (under the Theorem's assumptions). The time needed for the independently moving resources that are not ferries to reach the objective is

$$temp = dist(pl,po) / \min \{rsmr(i); i \varepsilon L\},$$

which equals +∞ if L is empty. Redefine dl:

(cont'd) because a single method that simultaneously permitted various speeds of carriers and various speeds of passengers would not admit a closed-form solution for dl and would be computationally burdensome.

$$dl \leftarrow \max \{dl, temp\}.$$

If $mode(i,s) \neq 1$ for some $i \in K$ or $i \in P$, go to Step 5(d). (The method of Step 5(c) is not used when some ferries or passengers are railroad rolling stock, aircraft, or watercraft.)

Step 5(c). Let

$$rsmr(i) = speed(i,s,kmp,pl,po)$$

for every $i \in P$. Let

$$V = \{i \in P: \ rsmr(i) > 0\}.$$

If the set V is empty, go to Step 5(d): the method of Step 5(c) is pointless unless some resources in P can move when not actually being carried. Let

$$W = \{i \in P: \ rsmr(i) = 0\}.$$

Let

$$veloc = \min \{rsmr(i);\ i \in C\}.$$

Define the weighted (harmonic) mean speed of the resources in V as

$$r = \frac{\sum_{i \in V} qrs(i)}{\sum_{i \in V} qrs(i) \,/\, rsmr(i)} \, .$$

If $r \geq veloc$, go to Step 5(d): the method does not apply.

The burden of the resources in V relative to load class lc is

$$burd2 = \sum_{i \in V} ldreq(i,lc,s) \ * \ qrs(i).$$

(Recall that $lc = fercl(kmp)$.) The burden that the resources in W place upon the resources in K may be alleviated because resources in V may be permitted to carry resources in W without increasing burd2.

Let

$$lcv = \max \{ldclas(i,s);\ i \in V\}.$$

If $lcv > 0$, then let

$$\hat{V} = \{i \in V: \ \mathit{ldclas}(i,s) = \mathrm{lcv}\},$$

$$\mathrm{kappa} = \sum_{i \in \hat{V}} \mathit{ldcap}(i,\mathrm{lcv},s) * \mathrm{qrs}(i),$$

$$\mathrm{beta} = \sum_{i \in W} \mathit{ldreq}(i,\mathrm{lcv},s) * \mathrm{qrs}(i),$$

$$\mathrm{temp} = \min\ \{\mathrm{kappa} / \mathrm{beta},\ 1\}.$$

If lcv = 0, let temp = 0. The burden that the resources in W place upon the resources in K is defined to be

$$\mathrm{burd1} = (1\text{-temp}) * \sum_{i \in W} \mathit{ldreq}(i,\mathrm{lc},s) * \mathrm{qrs}(i).$$

Let

$$\mathrm{alpha2} = (\mathrm{temp1} - \mathrm{temp2}) / (\mathrm{temp1} + \mathrm{temp3}),$$

where

$$\mathrm{temp1} = (r + \mathrm{veloc}) * \mathrm{burd2},$$
$$\mathrm{temp2} = 2 * r * \mathrm{burd1} * \mathrm{burd2} / \mathrm{cap},$$
$$\mathrm{temp3} = (\mathrm{veloc} - r) * \mathrm{burd1}.$$

The delay according to this method is

$$\mathrm{dc} = (\mathrm{dist}(\mathrm{pl},\mathrm{po}) / \mathrm{veloc}) * (a - 1),$$

where

$$a = 2*(r+\mathrm{veloc}) / (2*r + \mathrm{alpha2} * \mathrm{cap}/\mathrm{burd2} * (\mathrm{veloc}-r)).$$

Theorem 2 of Section 4.4 justifies this formula and clarifies the assumptions behind it; dist(pl,po) corresponds to the Proposition's D, cap corresponds to b, burd1 corresponds to f, burd2 corresponds to g, and veloc corresponds to s.

Step 5(d). Compute d1 as the lesser of the delay calculated by the method of Step 5(b) and the method of Step 5(c):

$$\mathrm{d1} \leftarrow \min\ \{\mathrm{dc},\ \mathrm{d1}\}.$$

Now d1 is the time needed for all the ferries and all the passengers to reach the objective (under the Theorem's assumptions). The time needed for the independently moving resources that are not ferries to reach the objective is

$$\mathrm{temp} = \mathrm{dist}(\mathrm{pl},\mathrm{po}) / \min\ \{\mathrm{rsmr}(i);\ i \in L\},$$

which equals $+\infty$ if L is empty. Redefine d1:

$$d1 \leftarrow \max \{d1, temp\}.$$

Step 6. Now d2, the delay attributable to barriers, is calculated. (At this point, d2 = 0, its initial value.) The task force can suffer a barrier delay only if some of its resources are traveling on land. Therefore, unless *mode*(i,s) = 1 or *mode*(i,s) = 3 for some i ε M, go to Step 7. If there is no movement barrier between cell pl and cell po--i.e., if movebar(pl,po) = 0--then go to Step 7.

Let

$$bt = movebar(pl,po).$$

If *barrr*(bt) = .true. and *mode*(i,s) = 3 for some i ε M, then let d2 = $+\infty$ and go to Step 7. If *mode*(i,s) $\neq$ 1 for every i ε M, go to Step 7.

Let I be the index set of every task force element with resources traveling under their own power on roads or cross-country:

$$I = \{i: \; 1 \le i \le n, \; [resources](u_i,j) > 0 \; \text{for some } j \; \varepsilon \; M \; \varepsilon \; mode(j,s) = 1\}.$$

(Recall that the task force elements are units $u_1,...,u_n$.) The task force's barrier delay is calculated as the greatest delay experienced by any of its elements in I:

$$d2 = \max \{bardly(\; butype(u_i), kmp, bt); \; i \; \varepsilon \; I\}.$$

Step 7. Compute d1= d0 + d + d2. End.

4.2 CALCULATING RESOURCES' SPEEDS

The function value speed(i,s,kmp,pl,po) is interpreted as the average speed at which side s type i resources can move, independently of other resources, from cell pl to cell po in movement posture kmp. It is derived according to the following procedure.

Step 0. Let

$$irs0 = irsoff(s).$$

Let

$$m = mode(i,s).$$

If m = 2, go to Step 2. In the remaining cases, the resources are traveling on the surface. They may only move to an adjacent cell: if cell po is not adjacent to cell pl, let speed = 0 and go to Step 5. If m = 1, go to Step 1. If m = 3, go to Step 3. If m = 4, go to Step 4.

Step 1. The type i resources are using some combination of road and cross-country movement. The rate at which they can move by road from cell pl to cell po is calculated as

$$\text{RMR} = \begin{cases} frd(\text{i,kmp,s})*vroll(\text{irs0+i,rt}) & \text{if rt} > 0 \text{ and } \text{i} \le \text{nequip(s)}, \\ vroll(\text{irs0+i,rt}) & \text{if rt} > 0 \text{ and } \text{i} > \text{nequip(s)}, \\ 0 & \text{if rt} = 0, \end{cases}$$

where rt = [road](pl,po), the type of road link between cells pl and po. (rt = 0 indicates no road link.) The game design datum *vroll*(irs0+i,k) is the basic road movement rate for side s type i resources on a type k road link; it should be 0 if, like supplies and some types of weapons, side s type i resources have no intrinsic movement capability. The design datum *frd*(i,kmp,s) is an adjustment factor applied to the basic movement rate when the task force to which the resources belong is in posture 29+kmp.

The design datum *vcc*(irs0+i,k) is the basic cross-country movement rate for side s type i resources in a type k environment; it should be 0 if they have no intrinsic movement capability or are completely road-bound. An adjustment factor, *fcc*(kmp,s), is applied when the resources belong to a task force in posture 29+kmp. The task force in question is assumed to travel half of the distance from cell pl to po in the environment of cell pl and the remaining half in the environment of cell po. The rate at which the type i resources can move cross-country in the environment of cell pl is

$$\text{CCMR1} = fcc(\text{kmp,s}) * vcc(\text{ irs0+i, [environment](pl)}),$$

and the rate at which they can move cross-country in the environment of cell po is

$$\text{CCMR2} = fcc(\text{kmp,s}) * vcc(\text{ irs0+i, [environment](po)}).$$

It remains to determine the fraction of distance traveled off-road (cross-country), denoted lambda. Of course, lambda = 1

if [road](pl,po) = 0. If, on the other hand, a road system links cell pl to cell po, lambda depends upon the type of road link, the extent to which it is damaged, and the movement posture.[1] Let

$$\text{phi} = \begin{cases} \textit{fccrd2}(\text{irs0+i}) * \textit{fccrd1}(\text{kmp,rt,s}) & \text{if rt} > 0, \\ 1 & \text{if rt} = 0, \end{cases}$$

where

$$\text{rt} = [\text{road}](\text{pl,po}).$$

The number phi is the fraction of total distance traveled that would be traveled off-road assuming that the road link, if any, were undamaged.[2] Let delta be the fraction of the road link between cells pl and po that is damaged, if one exists; Section 9 explains how this number is determined. Then

$$\text{lambda} = \begin{cases} \text{delta} + \text{phi} * (1\text{-delta}); & [\text{road}](\text{pl,po}) > 0 \\ 1; & [\text{road}](\text{pl,po}) = 0. \end{cases}$$

The formula relies on the assumption that resources must move cross-country wherever the roads are cut and elect to move cross-country elsewhere as often as they would if the entire road link were undamaged. Given the road movement rate, the cross-country movement rate in cell pl, the cross-country movement rate in cell po, and the fraction of distance traveled off-road, the resources' average speed is calculated as follows:

$$\text{speed} = \frac{2}{2 * \dfrac{1 - \text{lambda}}{\text{RMR}} + \dfrac{\text{lambda}}{\text{CCMR1}} + \dfrac{\text{lambda}}{\text{CCMR2}}};$$

0/0 is defined to be 0.

Go to Step 5.

---

[1]Dependence on posture is useful because, among other reasons it facilitates distinguishing between tactical movement and administrative movement. Dependence on road-link type is useful because, for example, the presence of several roughly parallel roads may allow the task force to deploy laterally, as in a tactical movement, although moving by road.

[2]The factor *fccrd*1(kmp,rt,s) accounts for the (interacting) effects of posture and road-link type on resources' propensity to travel off-road. The factor *fccrd*2(irs0+i) accounts for the basic propensity of type i resources to travel off-road; it is especially useful in the case of resources that cannot move effectively off-road, for which it should be 0 or almost 0.

Step 2. The resources are flying. Let

$$speed = vair(irs0+i).$$

Of course, this should be 0 if side s type i resources have no intrinsic flight capability. Go to Step 5.

Step 3. The resources are traveling by rail. If there is no rail link between cell pl and cell po--i.e., if [rail](pl,po) = 0--or if there is one but it is damaged, then let speed = 0 and go to Step 5. (Damage to rail links is explained in Section 9.) If there is an undamaged rail link between cell pl and po, let

$$speed = vroll(\ irs0+i,\ nroad + [rail](pl,po)).$$

Go to Step 5.

Step 4. The resources are sailing. For $1 \leq k \leq nenv$, the game design variable *ifsea*(k) should have the value .true. if and only if ships can sail in a type k environment. Therefore, if

$$ifsea([environment](po)) = .false.,$$

let speed = 0 and go to Step 5. If not, let

$$speed = vsea(irs0+i).$$

Step 5. End.

4.3 MOVEMENT DESIGN CONSIDERATIONS

The procedure for calculating the term of the movement delay proportional to the distance (the variable dl in Section 4.1) is not intended to be a detailed model of task force movement although its approach of viewing the task force as a collection of resources, rather than simply a collection of battle units, could accommodate greater detail. The model has no organic capability to impose rest periods and chaos on a moving task force's resources, and therefore the game design data on resource movement rates should already allow for rest and traffic delays. The model assumes that the distance resources travel in moving from one cell to another equals the straight line distance between the cells' centers, and therefore the design data on movement should allow for the inevitable indirectness of a route that conforms to terrain, roads, or railroads.

The procedure's mathematical complexity results from: (1) permitting a task force's more mobile resources to make more

trips than less mobile resources in the process of getting all the task force to its objective; (2) permitting slow-moving resources to benefit from high-speed transport without forcing them to stop moving while they wait for it. The first possibility is illustrated by trucks moving back and forth to take supplies, personnel, and towed artillery to the objective; the second is illustrated by personnel--specifically, infantrymen--marching when they are not riding, and carrying weapons and supplies whether marching or riding. The first possibility is implemented by Step 5(b), the second by Step 5(c), Section 4.1.

Because the procedure for calculating the movement delay considers the movement rates of the task force's various resources, it can discriminate the following effects:

- An infantry unit's movement rate can be increased by even a minor augmentation of trucks.
- A task force traveling cross-country in terrain unsuitable for wheeled vehicles can increase its movement rate by abandoning them or transferring them to other units.
- A task force's movement rate may vary with its resource composition, and not simply its aggregate ratio of transport capacity to demand.
- A task force may have diverse resources, some traveling on land while others fly.

4.3.1 Defining Load Classes and Movement Postures

Suppose the resources of side s (s = 1 or s = 2) consist of small arms, antitank weapons, artillery, tanks, APCs, wheeled vehicles, railroad rolling stock, aircraft, ships, supplies, and personnel. A reasonable definition of *ldclas*(*,s) is given below.

resource type	load class
small arms	0
antitank weapons	0
artillery	2
tanks	2
APCs	2
wheeled vehicles	3
R.R. rolling stock	5
aircraft	4
ships	6
supplies	0
personnel	1

Accordingly, *ferel* and *imel* might be defined as follows:

posture p	description	*ferel*(p-29)	*imelas*(p-29,*)
30	road/cross country tactical movement	3	2
31	road/cross-country administrative movement	3	2
32	rail movement	5	5
33	airborne movement	4	4
34	seaborne movement	6	6

Keeping the number of the highest load class and the number of the highest movement posture small conserves computer storage. So does listing the land movement postures before the air and sea movement postures and listing the road/cross-country movement postures before the rail movement postures.

With these values of *ldelas* and *ferel*, wheeled vehicles may make multiple trips back and forth, ferrying passengers to the objective, but tracked vehicles may not. In effect, IDAHEX would pool the wheeled vehicles of a task force in posture 30 or 31 and allocate them to personnel as well as weapons and supplies; consequently, an infantry unit might move faster than foot troops could walk. To get a task force containing infantry units to move at the pace of foot troops, one might introduce a dismounted march posture, but a simpler way would be to set *imelas*(30,i) and *imelas*(31,i) to 1 if a type i battle unit is an infantry unit.

If both sides lacked airlift and sealift capabilities, *ldelas*(*,s) and *ferel*, and *imelas* might be defined as follows:

resource type	load class
small arms	0
antitank weapons	0
artillery	2
tanks	2
APCs	2
wheeled vehicles	3
R.R. rolling stock	4
supplies	0
personnel	1

posture p	description	*ferel*(p-29)	*imelas*(p-29,*)
30	road/cross-country tactical movement	3	2
31	road/cross-country administrative movement	3	2
32	rail movement	4	4

Notice that resources with no intrinsic movement capability are put in load class 0. Personnel are put in load class 1, but they could be put in load class 0: then infantrymen would have to rely entirely on vehicles to transport themselves and their small arms--which would be acceptable if the forces are such that walking would be insignificant as a means of locomotion.

Putting artillery in the same load class as tanks implies that it is predominantly self-propelled. If it were predominantly towed, it should be put in class 0, and its burden on capacity in load class 2 (*ldreq*(i,2,s)) should be reduced to the extent that some of it is self-propelled. For greater resolution, artillery could be subdivided into self-propelled and towed.

4.3.2 Amphibious Operations

If *mode*(i,s) = 4, IDAHEX recognizes side s type i resources as ships. If *ifsea*(k) = .false., IDAHEX recognizes a type k environment as being land, rather than sea, and prohibits ships from entering it. To prevent land-bound resources from traveling at sea unless carried by ships, the game designer should make their cross-country movement rate 0 in sea cells; to be precise, *vcc* should be defined so that

$$vcc(\mathrm{irsoff}(s)+i,k) = 0$$

whenever $$ifsea(k) = .\mathrm{true}.$$

and $$mode(i,s) = 1 \text{ or } mode(i,s) = 3.$$

But if ships cannot enter a land cell and land-bound resources cannot move cross-country into a sea cell, how can a coastal cell be invaded or evacuated from the sea? The answer is to create one or more special types of road links, "roads to the sea", and to put one of these fictitious road links from each land cell where seaborne forces could land to an adjacent sea cell. If a type k road is a road to the sea, *vroll*(irsoff(s)+i,k) should be positive if and only if side s type i resources can use it to land from ships or to embark, and *fccrd*1(p-29,k,s) should be 0 for any $30 \leq p \leq 39$ such that posture p may be used for an amphibious landing.

4.4 THE MATHEMATICS OF TASK FORCE MOVEMENT

Theorems 1 and 2, below justify Steps 5(b) and 5(c), respectively, in the calculation of the movement delay. The reader can skip this section without losing continuity.

Theorems 1 and 2 rely on the following proposition.

Proposition. A positive quantity of buses and a quantity p of people ($p > 0$) are located at the same point. The buses and people must all move along a given route, whose length is D, to a given destination. They need not all depart at the same time nor all arrive at the same time. The total quantity of people that the buses can carry is K ($K > 0$). Buses and people are infinitely divisible. Let $\rho = K/p$. Assume that $\rho \leq 1$. Assume

(a) A bus moves, whether forward or backward along the route, at the constant speed s, where $s > 0$.

(b) Buses can load and unload people instantaneously.

(c) When not riding on a bus, people walk to the destination along the route at the constant speed r, where $s > r \geq 0$.

Then all the buses and people can be located at the destination after a length of time

$$T = \frac{D}{s}\left(\frac{2(r+s)}{2r + \rho(s-r)} - 1\right),$$

and no sooner.

Proof. First consider the case in which $r = 0$. Since buses are infinitely divisible, it is legitimate to treat them as n notional buses, n being a positive integer, each with capacity $\kappa = K/n$. Index the notional buses $1,\ldots,n$. A notional bus's journey consists of a sequence of segments in any of which it is moving either forward (to the destination) or backward along the route. Let f_i be the total distance notional bus i travels forward, and b_i the total distance it travels backward. For the bus to end its journey at the destination,

$$f_i - b_i = D.$$

Notional bus i completes its journey in a length of time equal to

$$\frac{1}{s}(f_i + b_i).$$

Since the people cannot walk ($r = 0$), they can all reach the destination only if

$$\kappa \sum_{i=1}^{n} f_i \geq pD.$$

Thus, a lower bound on T, the time needed to get all the buses and people to the destination, can be found by solving the mathematical programming problem

$$\text{minimize } \max \{\tfrac{1}{s}(f_i + b_i);\ 1 \le i \le n\}$$

$$\text{subject to } f_i - b_i = D \ \forall\ 1 \le i \le n$$

$$\kappa \sum_{i=1}^{n} f_i \ge \rho D$$

$$f_i,\ b_i \ge 0 \ \forall\ 1 \le i \le n.$$

An equivalent problem is

$$\text{minimize } \max \{f_i;\ 1 \le i \le n\}$$

$$\text{subject to } \sum_{i=1}^{n} f_i = \frac{n\rho D}{K}$$

$$f_i \ge 0 \ \forall\ 1 \le i \le n.$$

The solution is

$$f_i = \frac{pd}{K} = \frac{D}{\rho}$$

for each $1 \le i \le n$. Therefore, a lower bound on T is

$$T^L = \frac{1}{s}(f_i + f_i - D) = \frac{D}{s}\left(\frac{2}{\rho} - 1\right),$$

to which the Proposition's formula for T reduces when r = 0.

Now the bound must be shown to be attainable. Assume that ρ/κ is an integer. The approach is to construct a mathematically simple pattern of bus movements that attains the bound.[1]

Place markers at the origin, the destination, and uniformly along the route at intervals of D/n. Number them 0, 1,...,n,

[1]Other patterns attain the bound with much less frequent unloading and loading, and might be more representative of what would occur in practice, but they are more difficult to describe and analyze.

starting at the origin. For each $1 \leq i \leq n$, let bus i (now at the origin) load as many people as it can and proceed to marker i. Upon arriving at marker i, bus i executes the following procedure:

Step 1. If there are no people, in or out of buses, before marker i, go to Step 3.

Step 2. Go back to marker i-1, load as many people as possible, and then go forward to marker i. Go to Step 1.

Step 3. Go forward to marker n and stop.

Figure 4.1 illustrates the resulting pattern of bus movements in the case where $n = 3$, $p = 7$, and $\kappa = 1$.

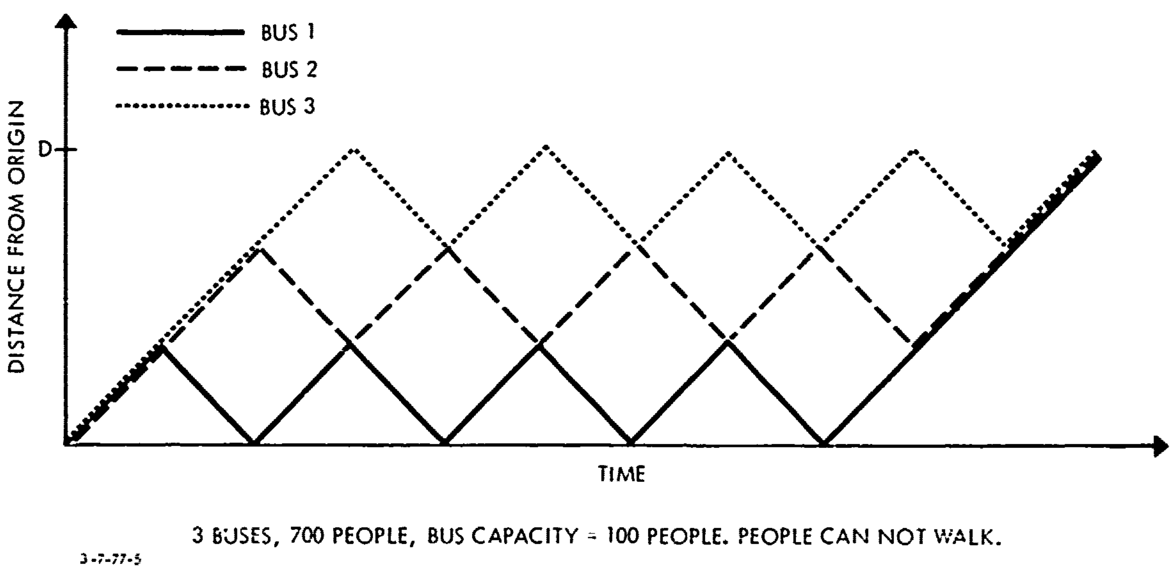


Figure 4.1. ILLUSTRATION OF BUS ROUTING PATTERN

Since p/κ is an integer, a notional bus is fully loaded whenever it travels forward. After the initial departure of all the notional buses from marker 0, the quantity of people left there is $p - n\kappa$. To transport them all to marker 1 or beyond in accordance with the above routing scheme, notional bus 1 must go back to marker 0 exactly

$$\frac{p - n\kappa}{\kappa}$$

times to pick up people, which implies

$$b_1 = \frac{p - n\kappa}{\kappa} \cdot \frac{D}{n} .$$

Moreover,

$$f_1 = D + b_1.$$

Therefore, notional bus i arrives at the destination and stops after the length of time

$$\frac{1}{s}(f_1 + b_1)$$

$$= \frac{1}{s}\left(D + 2D\left(\frac{p}{n\kappa} - 1\right)\right)$$

$$= \frac{D}{s}\left(\frac{2p}{n\kappa} - 1\right)$$

$$= \frac{D}{s}\left(\frac{2}{\rho} - 1\right) = T^L.$$

For any $i \geq 1$, the initial forward trip of notional bus $i + 1$ (to marker $i + 1$) is longer than the initial forward trip of notional bus i by the distance D/n, but the final forward trip of notional bus $i + 1$ (to marker n) is shorter than the final trip of notional bus i by the same distance. In between, their travels are mirror images; when notional bus $i + 1$ arrives back at marker i, it meets notional bus i. (See Figure 4.1.) Consequently, every notional bus arrives at the destination and stops at the same time, T^L.

That verifies the Proposition's conclusion in the case where $r = 0$, given the additional assumption that

$$\frac{p}{\kappa} = \frac{1}{\rho} \cdot n$$

is an integer. The verification is independent of the choice of n, and n can be chosen big enough so that p/κ is an integer if ρ is a rational number. Therefore, the expression for T is valid under the additional assumption that ρ is rational. Since the expression is continuous in ρ, and any real number can be approximated arbitrarily closely by a rational number, the conclusion is proved.

Now suppose $r > 0$. Create n notional buses as before. Define f_i and b_i as before. Since the people are infinitely divisible, they can be treated as m notional people, each of size $\sigma = p/m$. Index the notional people $1,\ldots,m$. Define w_j as the total distance that notional person j walks. Again,

$$f_i - b_i = D$$

for every $1 \le i \le n$. The total person-distance ridden plus the total person-distance walked must equal or exceed pD:

$$\kappa \sum_{i=1}^{n} f_i + \sigma \sum_{j=1}^{m} w_j \ge pD.$$

Notional bus i ends its journey in a length of time equal to

$$\frac{1}{s}(f_i + b_i),$$

and notional person j ends his journey in a length of time equal to

$$\frac{1}{r} w_j + \frac{1}{s}(D - w_j).$$

(A person never rides backward.) As before, it is easy to see that minimizing the time at which the last journey ends requires:

$$f_i = f_k, \; b_i = b_k, \; w_j = w_\ell,$$

$$\frac{1}{s}(f_i + b_i) = \frac{1}{r} w_j + \frac{1}{s}(D - w_j)$$

for every $1 \le i \le n$, $1 \le k \le n$, $1 \le j \le m$, $1 \le \ell \le m$. Thus, dropping the subscripts, the following equations must hold:

$$\frac{1}{s}(f + b) = \frac{1}{r} w + \frac{1}{s}(D - w)$$

$$n\kappa f + m\sigma w = pD$$

$$f - b = D.$$

Hence,

$$\frac{1}{s}(2f - D) = \left(\frac{1}{r} - \frac{1}{s}\right)w + \frac{D}{s}$$

$$= \left(\frac{1}{r} - \frac{1}{s}\right) \frac{pD - n\kappa f}{m\sigma} + \frac{D}{s} .$$

$$\Longrightarrow f = \frac{\frac{2D}{s} + \left(\frac{1}{r} - \frac{1}{s}\right)\frac{uc}{m\sigma}}{\frac{2}{s} + \frac{n\kappa}{m\sigma}\left(\frac{1}{r} - \frac{1}{s}\right)} = \frac{\frac{2D}{s} + \left(\frac{1}{r} - \frac{1}{s}\right)D}{\frac{2}{s} + \rho\left(\frac{1}{r} - \frac{1}{s}\right)} .$$

$$\Longrightarrow f = \frac{r + s}{2r + \rho(s - r)} D .$$

This implies that a lower bound on T is

$$T^L = \frac{1}{s}(f + b) = \frac{1}{s}(2f - D)$$

$$= \frac{D}{s}\left[\frac{2(r + s)}{2r + \rho(s-r)} - 1\right] = T.$$

The approach to verifying attainability of the lower bound is basically the same as before. But because, in the present case, people walk forward when they are not riding, bus i + 1 need not go back all the way to the point where bus i last unloaded. And because of this, the proper spacing of markers is harder to find. Let the distance between any two consecutive markers be δ; how to choose δ to attain a completion time equal to T is explained below. For the moment, one knows only that $\delta \leq D$. Let the notional buses follow the same routing scheme as before, but as soon as they begin their journeys, begin moving each marker forward at the rate r, to reflect dismounted people's walking. The number of trips backward that notional bus i makes ($1 \leq i \leq n$) is

$$\ell = \frac{p - n\kappa}{\kappa} ,$$

for the same reason as before--again assuming that p/κ is an integer. The distance of any trip forward from marker i - 1 to marker i can be easily calculated as

$$\frac{s\delta}{s-r} ,$$

and the distance of any trip backward--how far back the bus must go to reach marker i - 1 and, coincidentally, to intercept people walking--can be easily calculated as

$$\frac{s\delta}{r+s} \,.$$

The total distance traveled forward is given by

$$f = (\ell+n)\left(\frac{s\delta}{s-r}\right) ,$$

and the total distance traveled backward is given by

$$b = \ell\left(\frac{s\delta}{r+s}\right) .$$

The point reached when the journey ends is a distance

$$\pi = f - b = \left(\frac{(\ell+n)s}{s-r} - \frac{\ell s}{r+s}\right)\delta$$

from the origin. Of course, δ can be chosen so that $\pi = D$. The goal now is to show that if δ is so chosen, the length of time in which all buses and passengers complete their journeys equals T; $\pi = D$ if and only if

$$\delta = \frac{D}{\frac{(\ell+n)s}{s-r} - \frac{\ell s}{r+s}} .$$

Substituting in the expression for f reveals

$$f = \frac{D}{1 - \frac{\ell}{\ell+n} \cdot \frac{s-r}{r+s}}$$

$$= \frac{(\ell+n)(r+s)D}{2r\ell + nr + ns}$$

$$= \frac{\frac{p}{\kappa}(r+s)D}{2r(\frac{p}{\kappa}) - nr + ns}$$

$$= \frac{r+s}{2r + \rho(s-r)} D .$$

The last expression agrees with the value of f derived in finding the lower bound on T, T^L. Since $f - b = D$, the completion time achieved by the scheme equals T^L, which equals T.//.

<u>Theorem 1</u>. Buses and a quantity p of people ($p > 0$) are located at the same point; they must move along a given route, whose length is D, to a given destination. There are B differ-

ent types of buses, where $B > 1$. The total quantity of people that the type i buses ($1 \leq i \leq B$) can carry is K_i ($K_i > 0$).

Buses and people are infinitely divisible. Assume

$$\sum_{i=1}^{B} K_i \leq p.$$

Assume:

(a) For any $1 \leq i \leq B$, a type i bus moves, whether forward or backward along the route, at the constant speed s_i, where $s_i > 0$.

(b) Buses can load and unload people instantaneously.

(c) People never walk.

Then all the buses and people can be located at the destination after a length of time

$$T = \frac{D}{s_1}\left(\frac{2p_1}{K_1} - 1\right)$$

where

$$p_1 = \frac{p - \frac{1}{2}\sum_{i=2}^{B} K_i + \frac{1}{2s_1}\sum_{i=2}^{B} s_i K_i}{1 + \frac{1}{s_1 K_1}\sum_{i=2}^{B} s_i K_i},$$

and no sooner.

Proof. The essential problem is to assign people to buses. Let p_i be the total quantity of people assigned to buses of type i ($1 \leq i \leq B$). The minimum time in which the type i buses can get all these people and themselves to the destination is, by the Proposition,

$$T_i(p_i) = \frac{D}{s_i}\left(\frac{2p_i}{K_i} - 1\right).$$

(The quantity K_i/p_i corresponds to the Proposition's ρ.) Hence, the length of time which relocation of all people and buses can be completed equals

$$\max \{T_i(p_i);\ 1 \le i \le B\}.$$

This time is minimized if and only if

$$T_i(p_i) = T_1(p_1)$$

for every $1 \le i \le B$. This equation is equivalent to

$$p_i = \frac{s_i K_i}{2}\left(\frac{2}{s_1 K_1} p_1 + \frac{1}{s_i} - \frac{1}{s_1}\right).$$

Of course,

$$\sum_{i=1}^{B} p_i = p.$$

Substituting the expression for p_i in this equation for each $2 \le i \le B$ and solving for p_1 yields the conclusion's expression for p_1. And then

$$T = T_1(p_1).$$

<u>Theorem 2</u>. People and buses are located at the same point. They must move along a given route, whose length is D, to a given destination. There are two different types of people: the lazy and the stupid. The quantity of lazy people is p_1 ($p_1 > 0$), and the quantity of stupid people is p_2 ($p_2 > 0$). The total quantity of people that the buses can carry is K, where $K > 0$. Buses and people are infinitely divisible. Assume:

$$K \le p_1 + p_2.$$

Assume:

(a) A bus moves, whether forward or backward along the route, at the constant speed s, where $s > 0$.

(b) Buses can load and unload people instantaneously.

(c) Lazy people never walk. Stupid people walk toward the destination, along the given route, at the rate r, where $s > r \ge 0$.

Then all the buses and people can be located at the destination after a length of time

$$T = \frac{D}{s}\left(\frac{2(r+s)}{2r + (\alpha_2 K/p_2)(s-r)} - 1\right)$$

where

$$\alpha_2 = \frac{(r+s)p_2 - 2rp_1p_2/K}{(r+s)p_2 + (s-r)p_1} ,$$

and no sooner.

Proof. Let α_1 be the fraction of the buses assigned to transport lazy people (exclusively), and α_2 the fraction assigned to transport stupid people (exclusively); $\alpha_1 + \alpha_2 = 1$. The minimum relocation time for the lazy people and their buses is, by the Proposition,

$$T_1(\alpha_1) = \frac{D}{s}\left(\frac{2p_1}{\alpha_1 K} - 1\right) .$$

The minimum relocation time for the stupid people and their buses is, by the Proposition,

$$T_2(\alpha_2) = \frac{D}{s}\left(\frac{2(r+s)}{2r + (\alpha_2 K/p_2)(s-r)} - 1\right) .$$

The overall relocation time is minimized if and only if $T_1(p_1) = T_2(p_2)$. Equivalently,

$$\frac{2p_1}{\alpha_1 K} = \frac{2(r+s)}{2r + (\alpha_2 K/p_2)(s-r)} .$$

Substituting $1 - \alpha_2$ for α_1 and solving the above equation for α_2 yields

$$\alpha_2 = \frac{(r+s)Kp_2 - 2rp_1p_2}{(s-r)Kp_1 + (r+s)Kp_2}$$

And then $T = T_2(\alpha_2)$.

5. THE PRIMARY COMMANDS

At the start of each period, the Red player and the Blue player input commands to IDAHEX. A command is an instruction to battle units or a request for information. IDAHEX prevents a player from issuing instructions to enemy units or obtaining the enemy player's instructions to his units. The commands are fully described in the *Player's Manual*. This section discusses only the most important commands, which are all instructions to battle units.

5.1 ASSIGNING AND REVISING MISSIONS

Recall from Section 3.1 that a task force's change of status is always caused and directed by an order. A mission is a sequence of orders. Every task force has a mission, and every mission is assigned to exactly one task force (but two task forces may have identical missions). The same positive integer that identifies the task force identifies its mission. A mission's orders are stored in a pop-up stack, and are executed in sequence, from the top to the bottom. The order at the top of the stack is termed the "active order". If there is a start time associated with it, execution does not begin until the current time equals or exceeds the start time. When execution of the order is completed, it is removed from the stack, and the next order, if any, pops to the top.

A mission is created or modified by the mission command. If the player is modifying an existing mission, he identifies it by number and then lists the new orders in the sequence in which they are to be executed. The new orders completely replace the old orders. If the player is creating a new mission, he lists the orders and the elements of the task force (identified by their unit numbers). Creating the mission also creates the task force. When the mission ends, because it is accomplished or canceled, the task force ceases to exist as an organizational entity, and the number assigned to it and its mission becomes available for identifying a new task force and mission.

The following two examples are based on the area of war in Figure 3.3 (page 3-6) and the posture configurations assumed by Table 3.2 (page 3-8), namely:

npost(1) = 4, *npost*(2) = 1, *npost*(3) = 2, *npost*(4) = 3

pp	*pmapup*(pp)	*pmapdn*(pp)
10	20	-10
11	20	-10
12	21	-10
13	21	-10
14	21	-10
20	30	42
30	41	42
31	41	42
40	11	42
41	12	42
42	14	42

Example 1. Assume units 4 and 9, located in cell 17, are both in posture 12. In the following communications with IDAHEX, the Red player constitutes units 4 and 9 as a task force and assigns it a mission. Every line that IDAHEX writes on a player's terminal is preceded by a question mark to distinguish it. (IDAHEX does not actually write the question mark.) The player's replies are enclosed in quotation marks.

```
? Enter command.
  "mission"
? Enter orders.
  "16, 12, 0"
  "12, 30, 0"
  "12, 10, 0"
  ""
? List task force.
  "4,9"
```

Each of the three lines after the prompting phrase "Enter orders." states an order: the first number is the desired objective, the second the desired posture, and the third is the order's start time. The mission implies the following sequence of statuses for the task force consisting of units 4 and 9.

location	posture	objective
17	21	16
17	31	16
17	41	16
16	12	16
16	21	12
16	31	12
16	30	12
16	40	12
12	11	12
12	10	12

Example 2. Assume the posture class of unit 21 is 0. In the following communications with IDAHEX, the Blue player creates a mission for the task force consisting of unit 21:

```
? Enter command.
  "mission"
? Enter orders.
  "6, 13, 0"
  "9, 13, 0"
  ""
? List task force.
  "21"
```

The mission implies the following sequence of statuses for unit 21:

location	posture	objective
6	10	6
6	13	6
6	21	9
6	31	9
6	41	9
9	12	9
9	13	9

The example illustrates one way of accomplishing re-supply and replacement: if new resources should enter the area of war in cell i at time tr, the game design data should incorporate them into a unit whose initial location is i and initial posture class is 0, and then when t $\geq$ tr the player whose side should receive the resources can issue a mission command to activate the unit. An inactive unit first assumes posture 10 when it is activated (See Figure 3.1.)

Example 3. Assume the posture class of unit 21 is 0. In the following communications with IDAHEX, the Blue player activates unit 21 in cell 8 instead of its present location, cell 6:

```
? Enter command.
  "mission"
? Enter orders.
  "8, 0, 0"
  "8, 10, 0"
  ""
? List task force.
  "21"
```

The mission implies the following sequence of statuses for unit 21:

location	posture	objective
8	0	8
8	10	8

Thus, a player can activate one of his units in a cell different from its initial location; to do so, he must first change its location while it remains in posture class 0. This capability is necessary since the location where a package of supplies and replacements should become available might depend on the course of the game: in the first place, IDAHEX prohibits activation of a unit in a cell owned by the enemy or containing active enemy units; and it may be convenient to design the game so that supplies and replacements originate in corps, army, or front depots, which relocate to keep up with the combat forces, rather than fixed, theater depots. A player could use the capability to change inactive units' locations in order to cheat, activating units wherever he pleased; he might even order an active unit into posture class 0, order it to change location, and then re-activate it. Therefore, IDAHEX places an advisory message in the game designer's output file, file 51, whenever an inactive unit changes location.

In every example the mission's last order declares a hold posture as the desired posture. That is not essential because the player can always extend (modify) a mission some time after creating it. But he should avoid letting a task force complete its mission in a posture class other than -1, 0, or 1: to save time IDAHEX occasionally assumes that every disengaging, moving, or attacking unit belongs to a task force.

5.2 REDISTRIBUTING RESOURCES

One set of active units on the same side may transfer resources to another set of active, friendly units. The former set of units are called the givers; the latter set are called the takers. A unit may belong to both sets. A taker cannot receive resources it is prohibited from having--i.e., if unit j is a taker, it can receive type irs resources only if irs = *iars*(i,*butype*(j)) for some i--but subject to that restriction, it may receive any quantity. Any of three commands may be used to transfer resources: the send command, the transfer command, and the delivery command. The send command is the most powerful: it suffers only the minimal limitations noted above. The transfer and delivery commands suffer the following additional limitations: the givers and the takers must all have the same location; and the givers must all be in the transfer posture, posture *itrfp*. Since a player could use the send command to transfer resources instantaneously anywhere in the area of war, its use must be constrained by game rules devised by the game designer and imposed outside IDAHEX. As Section 8 explains, the send command is normally used only in a game in which the logistics systems are not played explicitly.

5.2.1 The Transfer Command

The transfer command causes an immediate, instantaneous transfer of resources from the givers to the takers. The command includes a list of the givers, a list of the takers, and the amount of each type of resource to be transferred from the set of givers to the set of takers. As an essential part of the command, the player declares the transfer location--the givers' and takers' location. If the player declines to furnish a list of givers, the list consists by default of every friendly unit whose location is the transfer location and whose posture is the transfer posture. If he fails to furnish a list of takers, the list consists by default of every active, friendly unit whose location is the transfer location and whose posture is not *itrfp*. If, despite the defaults, there are no givers or no takers, no transfer is made, and the player is warned. The player may also decline to declare the transfer amounts of one or more types of resources.

Let G be the set of givers, identified by their unit numbers, and T the set of takers, identified by their unit numbers. Let s = 1 if the units are Red and s = 2 if they are Blue. If G = T, then regardless of what transfer amounts the player specifies, all resources are pooled and then apportioned among the units. Assume G = T. Let $1 \leq irs \leq nrs(s)$. Define

$$T' = \{k \in T: \; iars(j, butype(k)) = \text{irs for some } 1 \leq j \leq nrst(butype(k))\}.$$

T′ is the set of takers that can have type irs resources.

Let

$$T'' = \{k \in T': \; toe(butype(k), \text{irs}) > 0\}.$$

If T′′ is nonempty, the resources of type irs are redistributed so that after redistribution

$$[\text{resources}](k,\text{irs}) \;/\; toe(butype(k),\text{irs})$$

is the same for every $k \in T''$ and [resources](k,irs) = 0 for every $k \in T - T''$.[1] Alternatively, if T′′ is empty, the resources are redistributed so that [resources](k,irs) is the same for every $k \in T'$ (and 0 for every $k \in T - T'$).

Henceforth, assume $G \neq T$. If, for any irs, the player does not declare the amount of type irs resources to be transferred, it is determined as

$$\min\{\text{demand}, \text{supply}\},$$

where

$$\text{demand} = \sum_{k \in T} \max\{toe(butype(k),\text{irs}) - [\text{resources}](k,\text{irs}), 0\}$$

and

$$\text{supply} = \sum_{k \in G} \max\{[\text{resources}](k,\text{irs}) - toe(butype(k),\text{irs}), 0\}.$$

That is, each taker demands the amount by which its stock falls short of its planned effective stock, and each giver offers the amount by which its stock exceeds its planned effective stock. If the player does declare the amount of type irs resources to be transferred, it is reset if necessary so that it does not exceed

$$\sum_{k \in G} [\text{resources}](k,\text{irs}),$$

the amount available. Let amt(irs) be the amount of type irs resources to be transferred.

---

[1] $T - T''$ is the set of every k such that $k \in T$ but $k \notin T''$.

The first step in accomplishing the transfer is allocating the resources among the takers. Let $1 \leq irs \leq nrs(s)$. Define T' and T'' as before. For any $k \in T$, let $q(k)$ be the quantity of type irs resources to be transferred to unit k, which must now be determined. If T' is empty, $q(k)$ is set to 0 for every k, and amt(irs) is reset to 0. Henceforth, assume T' is nonempty. Case 1: T'' is nonempty. Then the quantity amt(irs) is distributed among the battle units of T'' so as to equalize as much as possible their ratios of actual stock to planned effective stock. To be precise, $q(k)$ is set to 0 for every $k \in T - T''$, and $q(k)$ is chosen for every $k \in T''$ to

$$\text{minimize} \sum_{k \in T''} \left(1 - \frac{[\text{resources}](k,\text{irs}) + q(k)}{\textit{toe}(\textit{butype}(k),\text{irs})}\right)^2$$

$$\text{subject to} \sum_{k \in T''} q(k) = \text{amt}(\text{irs})$$

$$q(k) \geq 0 \text{ for every } k.$$

Alternatively, assume Case 2: T'' is empty. Then the quantity amt(irs) is distributed among the battle units of T' so as to equalize their stocks as much as possible. To be precise, $q(k)$ is set to 0 for every $k \in T - T'$, and $q(k)$ is chosen for every $k \in T'$ to

$$\text{minimize} \sum_{k \in T'} \Big([\text{resources}](k,\text{irs}) + q(k)\Big)^2$$

$$\text{subject to} \sum_{k \in T} q(k) = \text{amt}(\text{irs})$$

$$q(k) \geq 0 \text{ for every } k.$$

In both cases, after q is determined the transfer occurs: [resources](k,irs) is increased by the quantity $q(k)$ for each $k \in T$.

To complete the transfer, the givers must be assessed for the resources that have already been distributed to the takers. Let $1 \leq irs \leq nrs(s)$. For any $k \in G$, let $q(k)$ be the quantity of type irs resources to be taken from unit k, which must now be determined. Define

$$G' = \{k \in G: \textit{toe}(\textit{butype}(k),\text{irs}) = 0\}.$$

Let

$$Q = \sum_{k \in G'} [\text{resources}](k,\text{irs}).$$

For every $k \in G'$, $q(k)$ is set equal to

$$\min\{amt(irs)/Q,\ 1\} * [resources](k,irs).$$

(0/0 = 0 by convention.) If $Q \geq amt(irs)$, $q(k)$ is set to 0 for every $k \in G - G'$.

Otherwise, $q(k)$ is chosen for every $k \in G - G'$ to equalize as much as possible the units' ratios of actual stocks to planned effective stocks: $q(k)$ is chosen for every $k \in G - G'$ to

$$\text{minimize} \sum_{k \in G-G'} \left(\frac{[resources](k,irs) - q(k)}{toe(butype(k),irs)} - 1 \right)^2$$

$$\text{subject to} \sum_{k \in G-G'} q(k) = amt(irs) - Q$$

$$q(k) \geq 0 \text{ for every } k.$$

After q is determined the transfer occurs:

$$[resources](k,irs) \leftarrow [resources](k,irs) - q(k)$$

for every $k \in G$.

5.2.2 The Delivery Command

The delivery command allows the player to arrange a transfer of resources that will occur automatically, at the earliest possible moment. The command does this by creating a "delivery order" (not to be confused with the "orders" in a mission). A delivery order has four components: (1) the delivery task force; (2) the delivery destination; (3) the delivery size; (4) the intended recipients of the delivery. The delivery task force is the set of battle units intended to transfer resources to another set of units. The delivery destination is the cell where the delivery will occur. The delivery size is a number between 0 and 1, inclusive, that indicates how much should be transferred. The intended recipients must all belong to the player's side. The list of intended recipients may be empty. Once created, a delivery order continues to exist until the player cancels it or it is executed. Two or more delivery orders may name the same delivery task force, but if the delivery destinations are the same as well, confusion may result.

Suppose task force m has just entered posture *itrfp* in cell dd. IDAHEX must decide whether the transfer of resources will be governed by a transfer command that the player will issue later or by a delivery order. *IDAHEX infers that the player intends to issue a transfer command, and therefore makes no delivery of resources at this time, if either of the following conditions holds:*

(1) with this change of status, the task force has accomplished its mission;

(2) with this change of status, the task force has completed execution of its active order, and its new active order has a start time that exceeds the current time.

If neither condition holds, IDAHEX *searches for a delivery order --one whose delivery task force is* m *and delivery destination is* dd. *If none is found, a delivery order is generated, with the delivery task force* = m, *delivery destination* = dd, *delivery size* = 1.0, *and intended recipients* = *the empty set; a generated delivery order is treated as any other delivery order.*

Execution of the delivery order is a procedure very similar to the one initiated by a transfer command. Let G be the set of elements of task force m, identified by their unit numbers. G is the set of "givers". Let lambda be the delivery size, and let R be the set of intended recipients, identified by their unit numbers. If R is nonempty, let T be the set of every $k \in R$ such that unit k is active and located in cell dd; if R is empty, let T be the set of every active, friendly unit located in cell dd whose posture is not *itrfp*. T is the set of "takers". If T is empty, of course, no transfer occurs.

If G = T, then the resources are redistributed exactly as described for that case in Section 5.2.1.

Assume $G \neq T$. The amount of type i resources to be transferred is determined as

$$\min \{\text{demand}, \text{supply}\},$$

where

$$\text{demand} = \sum_{k \in T} \max \{\mathit{toe}(\mathit{butype}(k),i) - [\text{resources}](k,i), 0\}$$

and

$$\text{supply} = \text{lambda} * \sum_{k \in G} \max \{[\text{resources}](k,i) - \mathit{toe}(\mathit{butype}(k),i), 0\}.$$

Since the delivery size, lambda, may not exceed 1, a giver can only give away resources to the extent they exceed its planned effective stock. The first step in accomplishing the delivery is allocating the resources among the takers. The procedure is exactly the same as described in the previous subsection. The second step is assessing the givers for the resources that have

been distributed to the takers. The procedure is exactly the same as described in the previous subsection.

5.2.3 The Send Command

The send command is virtually identical in use and effect to the transfer command except that, since the participating units may all have different locations, it is not necessary to specify a transfer location. The command generates a list of givers and a list of takers. The player may specify the amount of any, or every, type of resources to be transferred. The actual quantity of a type of resources transferred from a giver to a taker is determined according to the same rules used to interpret the transfer command.

5.3 INITIATING SPECIAL ACTIVITIES

The activity command writes a line in the Special Activities list, creating either a close supporting fire activity (Section 6) or an LOC modification activity (Section 9). The special activity will continue until the player who set it up erases the line from the Special Activities List, through the cancel command, or until the task it defines is fully accomplished and IDAHEX cancels it.

6. COMBAT

As Section 3.4.2 explains, an engagement arises when units from one side attempt to occupy a cell containing enemy units in hold or disengagement postures. An engagement is not precipitated merely by a task force's entering an attack posture oriented toward a cell containing enemy units in hold or disengagement postures. The engagement arises when the task force attempts to change status from the attack posture oriented toward the enemy-owned cell to a hold posture in the enemy-owned cell.[1] The cell is termed the "engagement location". The force that precipitates the engagement, by attempting to occupy an enemy-owned cell, constitutes the engagement "attackers". Other friendly units may join the engagement later, possibly attacking from different locations; they, too, become "attackers". The enemy units whose location is the attacker's objective and whose postures are hold or disengagement constitute the engagement "defenders". Thus, at the outset of the engagement, one side is the attacker and the other side is the defender. These roles remain fixed throughout the engagement: even if the attackers succeed in occupying the engagement location, so that they are in hold postures and no longer attack postures, they are still the "attackers". An engagement ends when all its attackers have left or all its defenders have left. If an attacker's location is not the engagement location, it leaves its engagement when its objective becomes a cell other than the engagement location. If an attacker's location is the engagement location, it leaves its engagement when it enters a posture class other than 1 or 2. A defender leaves its engagement when it enters a posture class other than 1 or 2. Therefore, an attacker or defender leaves its engagement if it is destroyed (posture class -1).

Usually, a defender leaves its engagement by entering a movement posture.[2] While it is moving, the enemy cannot engage

[1] Sometimes, as Section 3.4.2 explains, the attempt by a task force in an attack posture to occupy its objective causes enemy units located there to revert to hold postures. After they have done so, it re-attempts occupation, precipitating an engagement.

[2] It is impossible for a unit to enter a movement posture oriented toward its own location.

it. That is one reason for the disengagement delay, and especially for making one term of the delay proportional to the anticipated movement delay. (See Section 3.2.3.) Loosely speaking, if the tactical situation implies that the unit is vulnerable to pursuit by engagement attackers, its disengagement delay (hence, the interval during which it is engaged) is extended to account for the combat with pursuing enemy units that its rearguard would have in reality.

Each engagement has a stylized FEBA that measures the attacker's progress. In any given engagement, the variable feba expresses the FEBA position as a fraction of *depth*. At the start of the engagement, feba = 0. At that point, all the attackers are in attack postures oriented toward the engagement location. If the attackers are sufficiently strong relative to the defenders, the FEBA advances--feba increases, to a maximum of 1. One might imagine that when feba is increasing the attackers are beating back the defenders; a more general, and more contemporary interpretation is that the attackers are penetrating the defender's formation. The game design datum *febad* is the criterion for deciding when the attackers have penetrated sufficiently to be allowed to occupy the engagement location. As soon as feba $\geq$ *febad*, ownership of the engagement location passes to the attacker's side, the attackers are allowed to enter the cell, and the defenders are forced to disengage and move out or be destroyed.

An engagement's FEBA is independent of other engagements' FEBAs and the general disposition of forces in the area of war. It may be interpreted as a measure of the attackers' penetration of the engagement location. But essentially it is just an abstraction used to determine how long the engagement lasts before the attackers defeat the defenders.

At the end of each frame, the results of every engagement during the frame are evaluated. If an engagement starts during a frame, the attackers can not possibly occupy the engagement location until the end of the frame, when the engagement's feba is updated. Therefore, *tframe* should be short enough to avoid delaying attackers excessively.

6.1 THE ATTRITION PROCESS

Attrition is essentially a Lanchester square process. The game design datum *katk*(i,j,k) is the quantity of enemy type j materiel disabled (destroyed or damaged) in one unit of time by a single side k type i ground-to-ground weapon belonging to an attacker, under the assumption that the side k weapon allocates all its fire to enemy type j materiel. The quantity disabled in one frame is

$$katk(i,j,k) = tframe * katk(i,j,k).$$

The datum *kdef*(i,j,k) is the quantity of enemy type j materiel disabled in one unit of time by a single side k type i ground-to-ground weapon belonging to a defender, under the assumption that the side k weapon allocates all its fire to enemy type j materiel. The quantity disabled in one frame is

$$kdef(i,j,k) = tframe * kdef(i,j,k).$$

Let cell loc be the engagement location of a given engagement. Let units $\{atk(i);\ 1 \le i \le natk\}$ be the attackers and units $\{def(i);\ 1 \le i \le ndef\}$ the defenders. Let sideA = 1 if the attackers are Red and sideA = 2 if they are Blue; let sideD = 3 - sideA. For each $1 \le i \le nrs(sideA)$, let

$$rsatk(i) = \sum_{k=1}^{natk} frinv(i,atk(k)) * [resources](atk(k),i),$$

the attackers' total quantity of type i resources that can become actively involved in combat or combat support. The function frinv is explicated in Section 5.4. Briefly, frinv(i,j) is the fraction of type i resources held by unit j that are available for combat, if the unit's type i resources are equipment, or that are available and needed for combat support, if its type i resources are support resources. For each $1 \le i \le nrs(sideD)$, let

$$rsdef(i) = \sum_{k=1}^{ndef} frinv(i,def(k)) * [resources](def(k),i),$$

the defenders' total quantity of type i resources that can become actively involved in combat or combat support. The current time, t, must coincide with the end of a frame. This subsection's goal is to derive the attrition suffered by each attacker and each defender during the frame just ended.

6.1.1 Kill Matrices and Potential Losses

Select an attacker-defender pair: for some $1 \le i \le natk$ and some $1 \le j \le ndef$, let

$$unitA = atk(i),\ unitD = def(j).$$

Of course, unitA is a positive integer identifying a battle unit. The phrase "battle unit unitA" is abbreviated below as simply "unitA". The phrase "battle unit unitD" is abbreviated as simply "unitD".

If one of unitA's type i ground-to-ground weapons ($1 \le i \le$ *nggwep*(sideA)) allocates all its fire to unitD's type j materiel ($1 \le j \le$ nmat(sideD)), the basic quantity of enemy type j materiel it disables in the frame is katk(i,j,sideA). But a weapon normally does not allocate all its fire to a single type of enemy materiel. This is not just a matter of doctrine. In reality, there might be several different types of materiel at which a weapon would fire; what it actually fired at would depend upon what targets it detected, and that would depend upon the composition and deployment of the enemy force. Two variables are used to adjust katk(i,j,sideA) for the allocation of fire--*stdtgt*(*,sideA) and *aggatk*(i,*,sideA). The game design datum *stdtgt*(j,sideA) is, for $1 \le j \le$ nmat(sideD), the quantity of type j materiel in a standard target force--a force representative of what side sideA might face in engagements. The design dataum *aggatk*(i,j,sideA) is the fraction of fire of a type i weapon from side sideA that is allocated to enemy type j materiel if the enemy materiel belongs to a standard target force. Let n = nmat(sideD). The fraction of fire of unitA's type i ground-to-ground weapons that is allocated to unitD's type j materiel is

alpha(i,j) =

$$\frac{\textit{aggatk}(i,j,\text{sideA}) * (\text{frinv}(j,\text{unitD}) * [\text{resources}](\text{unitD},j))}{\textit{stdtgt}(j,\text{sideA}) * \text{DEN}},$$

where

$$\text{DEN} = \sum_{k=1}^{n} \textit{aggatk}(i,k,\text{sideA}) * (\text{rsdef}(k) \,/\, \textit{stdtgt}(k,\text{sideA})).$$

The divisor DEN is just a normalizer, to ensure that the fractions of fire allocated to the various types of enemy materiel sum to 1.[1] Thus, if type j materiel is overrepresented compared with the standard target force, more fire is allocated to it; if no type j materiel is present, no fire is allocated to it.

The basic quantity of unitD's type j materiel that a single unitA type i weapon disables in the frame is

$$\text{katk}(i,j,\text{sideA}) * \text{alpha}(i,j).$$

---

[1]Because of this normalization, it is not necessary that

$$\sum_{j} \textit{aggatk}(i,j,\text{sideA}) = 1.$$

This quantity must be adjusted for the specific conditions of the engagement. Each adjustment affects either the lethality of unitA's type i weapons or the vulnerability of unitD's type j materiel to all enemy fire. In the former case, the adjustment takes the form of a factor applies to the row katk(i,*, sideA); in the latter, it takes the form of a factor applied to the column katk(*,j,sideA). The adjusted quantity of unitD's type j materiel that a unitA type i weapon disables in the frame is

$$\begin{aligned}\mathrm{K(i,j,unitA,unitD)} = \mathrm{katk(i,j,sideA)} &* \mathrm{alpha(i,j)}\\ &* \mathrm{PA} * \mathrm{PD}\\ &* \mathrm{EA} * \mathrm{ED}\\ &* \mathrm{B}\\ &* \mathrm{PREP}\end{aligned}$$

The sequel defines the factors.

Unit unitA must be in an attack posture unless its location coincides with the engagement location, in which case an attack posture must be extrapolated from its actual posture for the purpose of assessing its performance in the engagement. Let pa be unitA's posture, and define postA as follows:

$$\mathrm{postA} = \begin{cases}\mathrm{pa};\ \mathrm{pa} \geq 40\\ \mathit{pmapup}(\mathit{pmapup}(\mathrm{pa}));\ 20 \leq \mathrm{pa} \leq 29\\ \mathit{pmapup}(\mathit{pmapup}(\mathit{pmapup}(\mathrm{pa})));\ 10 \leq \mathrm{pa} \leq 19.\end{cases}$$

Let

$$\mathrm{PA} = [\mathrm{fckar}](\mathrm{i,sideA,postA}),$$

which equals *fckar*(i,sideA,*poff*(postA)) by definition. The factor PA adjusts the lethality of unitA's type i weapons according to unitA's attack posture. Let postD be unitD's posture. Let

$$\mathrm{PD} = [\mathrm{fckac}](\mathrm{j,sideD,postD}),$$

which equals *fckac*(j,sideD,*poff*(postD)) by definition. The factor PD adjusts the vulnerability of unitD's type j materiel according to unitD's defense posture.

Let e be the environment in the engagement location: e = [environment](loc). Let

$$\mathrm{EA} = \mathit{fckare}(\mathrm{i,sideA,e}).$$

The factor EA adjusts the lethality of unitA's type i weapons according to the environment in which the combat occurs. Let

$$ED = fckace(j,sideD,e).$$

The factor ED adjusts the vulnerability of unitD's type j materiel. The combat is tacitly assumed to occur in the engagement location; hence, the environment in unitA's location is irrelevant. This does not imply that the attacker benefits or suffers from terrain equally as the defender. The variables *fckare* and *fckace* provide factors that are applied only to katk(*,*,sideA), the attacker's "kill matrix"; other variables, namely *fckdre* and *fckdce*, provide factors that are applied to kdef(*,*,sideD), the defender's kill matrix.

Let bt be type of attack barrier between unitA's location and unitD's location: if cell locA is unitA's location, bt = atkbar(locA,loc). If bt = 0, let B = 1. (If there is no attack barrier, no adjustment is needed.)[1] If bt > 0 and

$$feba \le febab(bt)/depth,$$

let B = *fckarb*(i,sideA,bt); otherwise, let B = 1. Thus, even if a barrier exists, its effects cease when the attackers have progressed sufficiently.

The area of the "area of influence" of unit def(k) ($1 \le k \le ndef$) is zrarea(def(k)).[2] The area of the defenders' combined area of influence is computed as

$$defarea = \sum_{k=1}^{ndef} zrarea(def(k)).$$

The engagement variable front indicates the length of the defenders' line of contact with the attackers. Like FEBA, it is an abstraction, a way of measuring how far the defenders are stretched. If one or more attackers are located in cell loc, then front = $+\infty$. If not, the value of front depends on the number of directions from which the attack is coming. If the attackers are all located in the same cell, front equals the length of any side of a square equal in area to cell loc (a hexagon). If the attackers are located in k different cells, where k > 1, then front equals k times the length of any cell side. The depth of the defense, defdepth, is given by

$$defdepth = \min \{defarea/front, depth\}.$$

The defender's prepared positions, if any, are assumed to extend only to the depth defdepth. The factor PREP has two purposes:

---

[1]Recall from Section 2 that an attack barrier may exist in one direction but not the opposite direction.

[2]The function zrarea is explicated in Section 6.4 .

to reduce the vulnerability of a defender holding prepared positions, and to increase the vulnerability of a defender whose defense is hasty or disorganized. If unitD is not in a hold posture, let PREP = 1. Alternatively, assume that it is. The virtual length of time it has had to prepare its defense is t - tentry(unitD), which may be negative. (See Section 3.4.6.) Let

$$\text{pf} = \text{prep}\ (\text{j, sideD, t - tentry(unitD)}).$$

(The function prep is explicated in Section 6.4.) If pf < 1, unitD's preparation time is sufficient to reduce the vulnerability of its type j materiel provided it still holds prepared positions. Hence, let

$$\text{PREP} = \begin{cases} \text{pf if pf} \le 1 \text{ and feba} < \text{defdepth}/\textit{depth}, \\ 1 \text{ if pf} \le 1 \text{ and feba} \ge \text{defdepth}/\textit{depth}. \end{cases}$$

On the other hand, pf > 1 indicates a hasty, disorganized defense, a condition unlikely to improve just because the attackers progress. Hence, let

$$\text{PREP = pf if pf} > 1.$$

That completes derivation of K(i,j,unitA,unitD), the "potential" quantity of unitD's type j materiel ($1 \le j \le$ nmat(side)) disabled in the frame by a unitA type i ground-to-ground weapon ($1 \le i \le$ *nggwep*(sideA)). Similarly, the potential quantity of unitA's type j materiel ($1 \le j \le$ nmat(side A)) disabled in the frame by a unitD type i weapon ($1 \le i \le$ *nggwep*(sideD)) is

$$\text{K(i,j,unitD,unitA)} = \text{kdef(i,j,sideD)} * \text{alpha}'\text{(i,j)} * \text{PD}' * \text{PA}' * \text{ED}' * \text{EA}'.$$

The factors' definitions are analogous to those given above.

Let m = nmat(sideA). The allocation factor

$$\text{alpha}'(\text{i,j}) = \frac{\textit{aggdef}\text{(i,j,sideD)} * (\text{frinv(j,unitA)} * [\text{resources}]\text{(unitA,j))}}{\textit{stdtgt}\text{(j,sideD)} * \text{DEN}},$$

where

$$\text{DEN} = \sum_{k=1}^{m} \textit{aggdef}\text{(i,k,sideD)} * (\text{rsatk(k)} / \textit{stdtgt}\text{(k,sideD)}).$$

Let postD be unitD's posture. Let

$$PD' = [fckdr](i,sideD,postD),$$

which equals $fckdr(i,sideD,poff(postD))$ by definition. Define postA as before. Let

$$PA' = [fckdc](k,sideA,postA),$$

which equals $fckdc(j,sideA,poff(postA))$ by definition. Let e = [environment](loc). Let

$$ED' = fckdre(i,sideD,e),$$

$$EA' \quad fckdce(j,sideA,e).$$

That completes derivation of the two "potential kill matrices" for the attacker-defender pair unitA-unitD--K(*,*,unitA,unitD) and K(*,*,unitD,unitA). Potential kill matrices are derived for each attacker-defender pair.[1]

For any battle unit ibu and resource type irs, define

$$ERS(ibu,irs) = freff(ibu) * frinv(irs,ibu) * [resources](ibu,irs).$$

It is the effective quantity of the unit's type irs resources that can become actively involved in combat or combat support. The function freff is explicated in Section 6.4. Briefly, it adjusts a battle unit's effectiveness according to the density of friendly forces in its location. Let IGA = *nggwep*(sideA). Battle unit unitD's potential loss of type j materiel ($1 \le j \le nmat(sideD)$) in the frame due to all enemy ground fire is, by definition,

$$ploss(unitD,j) = \sum_{k=1}^{natk} \sum_{i=1}^{IGA} K(i,j,atk(k),unitD) * ERS(atk(k),i).$$

Let IGD = *nggwep*(sideD). Battle unit unitA's potential loss of type j materiel ($1 \le j \le nmat(sideA)$) in the frame due to all enemy ground fire is, by definition,

$$ploss(unitA,j) = \sum_{k=1}^{ndef} \sum_{i=1}^{IGD} K(i,j,def(k),unitA) * ERS(def(k),i).$$

---

[1]To conserve storage, the IDAHEX computer program uses none of these matrices. Of course, it gets the same results.

These potential losses reflect the interactions of the engaged units' weapons. In a theater-level game, in which the cell depth would ordinarily exceed the effective ranges of substantially all ground-to-ground weapons, those are the only units that could inflict losses. But in a corps-level game, some ground-to-ground weapons might be capable of firing effectively at units located two or more cells away. IDAHEX accommodates this as supporting fire, which is explained in Section 6.4. If ngn is IDAHEX's identification number of the engagement under consideration, the function call csf (ngn, sidea, esfatk) returns a vector esfatk for which esfatk(i) is the effective number of type i ground-to-ground weapons firing in support of the attackers; the function call spfire (ngn, sided, esfdef) returns a vector esfdef for which esfdef(i) is the effective number of type i ground-to-ground weapons firing in support of the defenders. Rather than using a parallel of this subsection's detailed procedure for determining weapons' effectiveness, the effectiveness of weapons in a supporting role is estimated as the average effectiveness of weapons of the same type participating directly in the engagement.[1] These average effectiveness numbers are the entries in the two kill matrices, A and D, derived below.

Let $1 \leq i \leq IGA$ and $1 \leq j \leq nmat(sideD)$. Let $1 \leq k \leq natk$. The total potential loss of enemy type j materiel attributed to all the type i weapons of attacker k equals

$$\sum_{\ell=1}^{ndef} K(i,j,atk(k),def(\ell)) * ERS(atk(k),i).$$

The formula commits no double-counting because the array K takes into account the allocation of type i weapons' fire to the various types of materiel in the various enemy units. The total potential loss of enemy type j materiel attributed to all the attackers' type i weapons equals

$$\sum_{k=1}^{natk} \sum_{\ell=1}^{ndef} K(i,j,atk(k),def(\ell)) * ERS(atk(k),i).$$

Therefore, the average potential loss of enemy type j materiel attributed to a type i weapon that is effectively, actively involved in combat is

---

[1]This does not mean that range is ignored. The possibly greater range of supporting weapons from targets is considered in the calculation of esftk and esfdef.

$$A(i,j) = \frac{\sum_{k=1}^{natk} \sum_{\ell=1}^{ndef} K(i,j,atk(k),def(\ell)) * ERS(atk(k),i)}{\sum_{k=1}^{natk} ERS(atk(k),i)} .$$

The matrix A is an "average kill matrix" for the attackers as a whole. The defenders' average kill matrix D, is defined analogously: for $1 \le i \le IGD$ and $1 \le j \le nmat(sideA)$,

$$D(i,j) = \frac{\sum_{k=1}^{ndef} \sum_{\ell=1}^{natk} K(i,j,def(k),atk(\ell)) * ERS(def(k),i)}{\sum_{k=1}^{ndef} ERS(def(k),i)} .$$

Let $1 \le j \le nmat(sideD)$. The defenders' total potential loss of type j materiel due to the attackers' supporting fire is determined as

$$sigma = \sum_{i=1}^{IGA} A(i,j) * esfatk(i).$$

The loss is distributed among the defenders in proportion to their potential losses from the attackers' own fire: the loss borne by defender k is given by

$$x\ell(k) = \frac{ploss(def(k),j)}{\sum_{\ell=1}^{ndef} ploss(def(\ell),j)} * sigma .$$

After finding $x\ell(k)$ for every $1 \le k \le ndef$, ploss is redefined:

$$ploss\ (def(k),j) \leftarrow ploss(def(k),j) + x\ell(k)$$

for every $1 \le k \le ndef$.

Let $1 \leq j \leq$ nmat(sideA). The attackers' total potential loss of type j materiel due to the defenders' supporting fire is determined as

$$\text{sigma} = \sum_{i=1}^{IGD} D(i,j) * \text{esfdef}(i).$$

The procedure for redefining ploss to reflect the attackers' losses of materiel due to enemy support fire is analogous to the one above.

The matrices A and D are passed to the subprogram app for use in the antipotential potential method. In that context, a theoretically rigorous approach would create A and D not by averaging (as above) but by using artificial weapon types. Unless

$$K(i,j,atk(k'),def(\ell)) = K(i,j,atk(k''),def(\ell))$$

and

$$K(j,i,def(\ell),atk(k')) = K(j,i,def(\ell),atk(k''))$$

for every $1 \leq j \leq$ IGD and $1 \leq \ell \leq$ ndef, it would re-classify type i weapons belonging to attacker k′ and type i weapons belonging to attacker k″ as two different types of weapons. And unless

$$K(i,j,def(k'),atk(\ell)) = K(i,j,def(k''),atk(\ell))$$

and

$$K(j,i,atk(\ell),def(k')) = K(j,i,atk(\ell),def(k''))$$

for every $1 \leq j \leq$ IGA and $1 \leq \ell \leq$ natk, it would re-classify type i weapons belonging to defender k′ and type i weapons belonging to defender k″ as two different types of weapons. Corresponding to an increase in the number of different types of weapons the attackers and defenders had would be an increase in the number of rows and columns of A and D. The matrices might grow so large that they required too much main storage and led to excessive execution times for app.

6.1.2 Weapons' Values

The antipotential potential method finds consistent values (antipotential potentials) for weapons based on the rates at

which they kill (or, more accurately, disable) enemy weapons. It was discovered independently by Spudich [6] (also see [7]), by Dare and James [3], and by Thrall and Howes [5]. Their work was synthesized by Anderson [2]. The IDAHEX subprogram app determines the value of each type of weapon in a given engagement. The present version of app computes these values from the kill matrices A and D, derived in Section 6.1.1, by Holter's version of the antipotential potential method [4].

Recall that A(i,j) is the (average) rate at which a type i ground-to-ground weapon belonging to the attackers kills (strictly speaking, disables) the defenders's type j materiel and D(i,j) is the (average) rate at which a type i ground-to-ground weapon belonging to the defenders kills the attackers' type j materiel. The antipotential potential method deals only with ground-to-ground weapon interactions, and therefore uses only the submatrices of A and D relevant to ground-to-ground weapons killing ground-to-ground weapons. In this subsection, the symbols A and D represent those submatrices. Let

$$m = \mathit{nggwep}(\text{sideA}),\ n = \mathit{nggwep}(\text{sideD}).$$

The matrix A is $m \times n$, and D is $n \times m$. Let wa be the m-vector whose i-th component is the amount of type i weapons held by the attackers, and let wd be the n-vector whose j-th component is the amount of type j weapons held by the defenders. Let va be an m-vector and vd an n-vector. The component va(i) $(1 \le i \le m)$ is the value of a type i weapon belonging to the attackers, and vd(j) $(1 \le j \le n)$ is the value of a type j weapon belonging to the defenders; the values are derived below.

Some notation is needed. Suppose v and w are real s-vectors, and M is a real $r \times s$ matrix. Then

$$\langle v, w \rangle = \sum_{i=1}^{s} v(i) * w(i),$$

and M * v is the r-vector whose i-th component equals

$$\sum_{j=1}^{s} M(i,j) * v(j).$$

(Unless noted otherwise, every vector is a column vector.) The transpose of M is denoted "M^t": $M^t(i,j) = M(j,i)$ for every $1 \le i \le r$ and $1 \le j \le s$.

The antipotential potential method defines va and vd so that, for some scalar alpha,

$$(1) \quad alpha * va(i) = \sum_{j=1}^{n} A(i,j) * vd(j) \text{ for every } 1 \le i \le m$$

and, for some scalar delta,

$$(2) \quad delta * vd(i) = \sum_{j=1}^{m} D(i,j) * va(j) \text{ for every } 1 \le i \le n.$$

Thus, each weapon's value is proportional to the rate at which it destroys enemy value. By equation (2),

$$vd(j) = (1/delta) * \sum_{k=1}^{m} D(j,k) * va(k).$$

Substitute for vd in equation (1), to conclude

$$(3) \quad (alpha * delta) * va(i)$$

$$= \sum_{k=1}^{m} \sum_{j=1}^{n} A(i,j) * D(j,k) * va(k)$$

$$= \sum_{k=1}^{m} AD(i,k) * va(k),$$

where AD is the matrix product of A and D. Let

$$lambda = alpha * delta.$$

Equation (3) says that va is an eigenvector of the matrix AD and lambda is an eigenvalue. According to the Frobenius Theorem, if AD is nonnegative and irreducible, then equation (3) has a solution in which lambda > 0 and va ≥ 0, and such a solution is unique up to multiplication of va by a positive scalar. Of course, AD is nonnegative. The matrix AD is "irreducible" if and only if it is not "reducible". By definition, AD is reducible if and only if re-ordering its rows and columns can put it in the form

$$\left[\begin{array}{c|c} M1 & 0 \\ \hline M3 & M2 \end{array}\right],$$

where M1 and M2 are square matrices and all the elements in the upper right-hand block are zero. Permuting the rows and columns of AD is equivalent to permuting the rows of A and the columns of

D before calculating the product matrix. It follows that the nonnegative matrix AD is reducible if and only if there are non-emply subset A1 and A 2 of the set $\{i: 1 \leq i \leq m\}$ such that: the number of elements in A2 equals m minus the number of elements in A1; and if $A(i,j) > 0$ for some $i \in A1$ and $1 \leq j \leq n$, then $D(j,k) = 0$ for every $k \in A2$. The condition holds if, for example, the attackers' weapons of a certain tyre are invulnerable to the defenders' fire.[1] Thrall argues that the weapon values obtained by the antipotential potential method are meaningful even if AD is reducible [5].[2]

Several ways of scaling va and resolving lambda into the factors alpha and delta have been proposed. Each of the following sets of assumptions uniquely determines va (determines how it should be scaled) and alpha and delta:

(i) $\sum_{i=1}^{n} va(i) = 1, \sum_{i=1}^{n} vd(i) = 1$ (Dare and James)

(ii) $delta = \sum_{i=1}^{m} va(i), \; alpha = \sum_{i=1}^{n} vd(i)$ (Thrall and Howes)

(iii) $delta = \langle va,wa \rangle , \; alpha = \langle vd,wd \rangle$ (Spudich in TATAWS III)

(iv) alpha = delta, va(kw) = 1 (Holter).

In (iv) kw is an integer in the interval [1,m]. The requirement va(kw) = 1 merely fixes the scaling of va; choice of kw does not affect the relative proportions of the elements of va and va.[3] The present version of app implements (iv).

---

[1]The matrix AD is reducible if D(*,j) = 0 for some j, which is necessarily true (because of the allocation of fire) if the attackers have no type j weapons. IDAHEX circumvents this problem by working, in effect, with an irreducible submatrix of AD.

[2]His argument assumes that the antipotential potential method finds the weapon value by an iterative procedure starting with all-positive values. Such is the app procedure.

[3]Some antipotential potentials may be 0. Of course, if va(kw) = 0, no rescaling can make va(kw) = 1. IDAHEX's subprogram app chooses kw to avoid this contradiction if possible. The contradiction is avoidable unless the only nonnegative solution of equations (1) and (2) is alpha = delta = 0, va = 0, vd = 0.

For arguments in favor of scaling assumption (iv) and against the three alternatives, see [4]. The primary consideration in selecting a scaling assumption is the reasonableness of the resulting force ratio:

$$FR = \frac{\langle va, wa \rangle}{\langle vd, wd \rangle} .$$

It should indicate which side is dominant. The attackers are said to dominate if the force ratio rises as combat continues. That happens if and only if the defenders' rate of value loss is bigger in proportion to their total value than the attackers'--i.e., the quantity

$$FR2 = \left(\frac{\langle vd, A^t * wa \rangle}{\langle vd, wd \rangle} \right) \Big/ \left(\frac{\langle va, D^t * wd \rangle}{\langle va, wa \rangle} \right)$$

exceeds 1. But

$$FR2 = \frac{\langle A*vd, wa \rangle}{\langle D*va, wd \rangle} \cdot \frac{\langle va, wa \rangle}{\langle vd, wd \rangle}$$

$$= \frac{alpha * (\langle va, wa \rangle)^2}{delta * (\langle vd, wd \rangle)^2} .$$

The first of the two preceding equalities reveals that the value of FR2 is independent of how va and vd are scaled. Under scaling assumption (iv), the force ratio, FR, equals the square root of FR2 (and therefore exceeds 1 if and only if FR2 exceeds 1). Under assumptions (i) and (ii), it is possible that FR > 1 while FR2 < 1, and *vice versa*. Under assumption (iii), FR > 1 if and only if FR2 > 1, but regardless of the force ratio the attackers lose value at the same rate as the defenders:

$$\begin{aligned}
\langle va, D^t * wd \rangle &= \langle D*va, wd \rangle \\
&= delta * \langle vd, wd \rangle \\
&= \langle va, wa \rangle * alpha \\
&= \langle A*vd, wa \rangle \\
&= \langle vd, A^t * wa \rangle .
\end{aligned}$$

Hence, assumption (iv) appears to be the most suitable.

The subprogram app actually determines the value of every resource, not just ground-to-ground weapons. Let mm = nrs(sideA) and nn = nrs(sideD). Let

$$\text{vala}(i) = \begin{cases} \text{va}(i); & 1 \le i \le m \\ 0\quad ; & m < i \le mm \end{cases}$$

$$\text{vald}(i) = \begin{cases} \text{vd}(i); & 1 \le i \le n \\ 0\quad ; & n < i \le nn \end{cases}$$

Since the resources other than ground-to-ground weapons cannot destroy enemy resources, giving them zero value is completely consistent with the antipotential potential method. Indeed, one might expand A and D to include all resource types, so A would have mm rows and nn columns and D would have nn rows and mm columns. Of course, A(i,j) would be 0 unless $i \le m$, and D(i,j) would be 0 unless $i \le n$. The vectors vala and vald defined above would satisfy equations (1) and (2) using the expanded A and D:

$$\text{alpha} * \text{vala}(i) = \sum_{j=1}^{n} A(i,j) * \text{vald}(j) + \sum_{j=n+1}^{nn} A(i,j) * \text{vald}(j)$$

for every $1 \le i \le mm$, and

$$\text{delta} * \text{vald}(i) = \sum_{j=1}^{m} D(i,j) * \text{vald}(j) + \sum_{j=m+1}^{mm} D(i,j) * \text{vald}(j)$$

for every $1 \le i \le nn$.

6.1.3 Actual Losses of Materiel

Section 6.1.1 derives the potential losses of materiel suffered by each battle unit in the given engagement. Section 6.1.2 derives the values of the resources in the engagement. Those subsections' notation remains in force. Recall that ERS(ibu,i) is the effective quantity of type i resources belonging to battle unit ibu that can become actively involved in combat or combat support; also recall that esfatk(i) is the effective number of type i ground-to-ground weapons firing in

support of the attackers, and esfdef(i) is the effective number of type i ground-to-ground weapons supporting the defenders. Let

$$
\text{ersatk}(i) = \begin{cases} \displaystyle\sum_{k=1}^{\text{natk}} \text{ERS}(\text{atk}(k),i) + \text{esfatk}(i); \; 1 \le i \le \text{IGA} \\ \displaystyle\sum_{k=1}^{\text{natk}} \text{ERS}(\text{atk}(k),i); \; \text{IGA} < i \le \text{nrs}(\text{sideA}) \end{cases}
$$

$$
\text{ersdef}(i) = \begin{cases} \displaystyle\sum_{k=1}^{\text{ndef}} \text{ERS}(\text{def}(k),i) + \text{esfdef}(i); \; 1 \le i \le \text{IGD} \\ \displaystyle\sum_{k=1}^{\text{ndef}} \text{ERS}(\text{def}(k),i); \; \text{IGD} < i \le \text{nrs}(\text{sideD}) \end{cases}
$$

$$m = \text{nmat}(\text{sideA}),$$

$$n = \text{nmat}(\text{sideD}).$$

The attackers' total value, fgrd, is defined by

$$\text{fgrd} = \sum_{i=1}^{m} \text{ersatk}(i) * \text{vala}(i),$$

and the defenders' total value, ggrd, is defined by

$$\text{ggrd} = \sum_{i=1}^{n} \text{ersdef}(i) * \text{vald}(i).$$

The engagement's ground force ratio is

$$\text{FRGRD} = \text{fgrd} / \text{ggrd}.$$

The calculation of the family of kill matrices K(*,*,atk(k),def(ℓ)), the average kill matrices (A and D), and the potential losses (ploss) considered several influences, listed in Table 6.1. But the values assigned to the relevant variables by the game design data may not adequately represent

Table 6.1. INFLUENCES ON ATTRITION

Influence	How Represented
attack *vs.* defense	katk *vs.* kdef
posture	*fckar, fckac, fckdr, fckdc*
environment	*fckare, fckace, fckdre, fckdce*
barriers	*fckarb*
defensive preparation	prep

all these influences, necessitating adjustments to ploss. In addition, ploss must be scaled according to the intensity of combat. Finally, no unit should be assessed losses in excess of what it has.

The first step in the adjustment process is determining a representative posture for the engagement attackers and one for the defenders. In accordance with Section 6.1.1, if an attacker's actual posture is pa, its representative attack posture is

$$\text{postA} = \begin{cases} \text{pa};\ \text{pa} \geq 40 \\ \mathit{pmapup}(\mathit{pmapup}(\text{pa}));\ 20 \leq \text{pa} \leq 29 \\ \mathit{pmapup}(\mathit{pmapup}(\mathit{pmapup}(\text{pa})));\ 10 \leq \text{pa} \leq 19. \end{cases}$$

Let postA be that attack posture such that the total value of the attackers for which it is the representative attack posture is greatest. The value of attacker k is, by definition,

$$\sum_{i=1}^{mm} \text{ERS}(\text{atk}(k),i) * \text{vala}(i).$$

The next step compares the attackers' value loss implied by ploss with the value loss prescribed by the engagement's force ratio.[1] The attackers' potential loss of value is

[1]This step is basically the same as one in IDAGAM's attrition procedure [1]. Indeed, the basic structure of IDAHEX's attrition procedure--a scaled Lanchester square process--originated with IDAGAM.

$$\text{delval} = \sum_{k=1}^{\text{natk}} \sum_{i=1}^{m} \text{ploss}(\text{atk}(k),i) * \text{vala}(i).$$

Let temp = frdval(FRGRD,postA). (The function frdval is explicated in Section 6.4.) *If* temp < 0, *this step is skipped.* Thus, by appropriately defining the game design data used by frdval, the game designer can selectively avert this step. If temp $\geq$ 0, let

$$\text{scalar} = \text{temp} \;/\; (\text{delval}/\text{fgrd}),$$

and redefine ploss: for every $1 \leq k \leq$ natk and $1 \leq$ irs $\leq m$,

$$\text{ploss}(\text{atk}(k),\text{irs}) \longleftarrow \text{scalar} * \text{ploss}(\text{atk}(k),\text{irs}).$$

That is, the attackers' potential losses are scaled so that the attackers' total potential loss of value agrees with what is predicted from the force ratio.

Next, the same operation occurs for the defenders. Let

$$\text{delval} = \sum_{k=1}^{\text{ndef}} \sum_{i=1}^{n} \text{ploss}(\text{def}(k),i) * \text{vald}(i).$$

Let temp = frdval(FRGRD,postD). *If* temp < 0, *this step is skipped.* Otherwise, let

$$\text{scalar} = \text{temp} \;/\; (\text{delval}/\text{ggrd}),$$

and redefine ploss: for every $1 \leq k \leq$ ndef and $1 \leq$ irs $\leq n$,

$$\text{ploss}(\text{def}(k),\text{irs}) \longleftarrow \text{scalar} * \text{ploss}(\text{def}(k),\text{irs}).$$

The final step is scaling ploss according to the intensity of combat, which depends upon the tactical overlap of the attacking force and the defending force. The tactical overlap is defined as the depth of the attackers' penetration of the defenders' cell (feba * *depth*) plus the effective range of the attackers' fire, which depends upon the combat environment. To be precise, the tactical overlap is defined as

TO = min {feba * *depth* + *td*([environment](loc)), defdepth}.

(Recall that cell loc is the engagement location, and defdepth, defined in Section 6.1.1, is the depth of the defense.) The intensity of combat is indicated by

$$TI = TO \;/\; defdepth,$$

a number between 0 and 1. If the attackers and defenders are colocated, then feba = 1, and TI = 1. The potential losses are scaled by TI:

$$ploss(atk(k),irs) \longleftarrow TI * ploss(atk(k),irs),$$

for every $1 \le k \le natk$ and $1 \le irs \le m$, and

$$ploss(def(k),irs) \longleftarrow TI * ploss(def(k),irs)$$

for every $1 \le k \le ndef$ and $1 \le irs \le n$.

The losses can now be assessed. Usually, a unit can only lose resources that are actively involved in combat. If FRGRD $\ge .0001$, then for every $1 \le k \le natk$, [resources](atk(k),irs) is reduced by the quantity

$$loss(atk(k),irs) = \min \{ploss(atk(k),irs),\; frinv(irs,atk(k)) * [resources](atk(k),irs)\}$$

for every $1 \le irs \le m$. But if FRGRD $< .0001$, the attackers lose everything: for every $1 \le k \le natk$

$$[resources](atk(k),irs) \longleftarrow 0$$

for every $1 \le irs \le nrs(sideA)$. That eliminates the possibility of dummy attackers, in which the attackers have no ground-to ground weapons available for combat and suffer no losses. If FRGRD $\le 10{,}000$, then for every $1 \le k \le ndef$, [resources](def(k),irs) is reduced by the quantity

$$loss(def(k),irs) = mi\ \{ploss(def(k),irs),\; frinv(irs,def(k)) * [resources](def(k),irs)\}$$

for every $1 \le irs \le n$. If FRGRD $> 10{,}000$, then for every $1 \le k \le ndef$

$$[resources](def(k),irs) \longleftarrow 0$$

for every $1 \le irs \le nrs(sideD)$.

6.1.4 Losses of Personnel

In the extreme cases where the ground force ratio is very small or very large, one side loses all its resources and the other side loses none, as the preceding subsection explains. In the other case, the actual losses of materiel, found in the preceding section, and recorded in the variable loss, imply losses of personnel. The game design datum *dpersr*(p.i.j) is defined as a Red battle unit's loss of type p personnel associated with the loss of a unit-quantity of type j materiel to fire from a Blue type i weapon; *dpersb* is the analogous datum for a Blue battle unit. Let unit unitD be one of the defenders. Let nm = nmat(sideD) and IGA = *nggwep*(sideA). The fraction of unitD' losses of type j materiel attributable to enemy type i weapon is compuated as

$$\alpha(i,j) = \frac{A(i,j) * \text{ersatk}(i)}{\sum_{k=1}^{IGA} A(k,j) * \text{ersatk}(k)} .$$

The actual losses of unitD's type j materiel attributable to the attackers' type i weapons is therefore

$$\alpha(i,j) * \text{loss}(\text{unitD},j).$$

Consequently, unitD's loss of type p personnel ($1 \le p \le$ *npers*(sideD)) as a result of the attackers' fire is

$$\text{ploss}(\text{unitD},\text{nm}+p) =$$

$$\sum_{j=1}^{nm} \sum_{i=1}^{IGA} dpersr(p,i,j) * (\alpha(i,j) * \text{loss}(\text{unitD},j))$$

if unitD is Red; *dpersb* replaces *dpersr* if unitD is Blue. Unit unitD's actual loss of type p personnel is

$$\text{loss}(\text{unitD},\text{nm}+p) =$$

$$\min \{\text{ploss}(\text{unitD},\text{nm}+p), [\text{resources}](\text{unitD},\text{nm}+p)\},$$

and [resources](unitD,nm+p) is reduced by this quantity. An attackers' actual losses of personnel are assessed analogously (using the matrix D instead of A, and ersdef instead of ersatk).

6.1.5 Repairable Equipment

The actual losses of materiel, found in Section 6.1.1 and recorded in the variable loss, may include equipment that is merely damaged. Equipment that can be repaired must be placed in one of two repair classes. The array *frdmg* gives the fraction of disabled equipment that goes to each repair class given the type of enemy weapon that disabled it. However, *frdmg* is not defined and no repairs are made if the value of the logical variable *ifrep* is .false..

Although equipment might be repaired in depots serving many units, IDAHEX segregates the unit's equipment so that it can return repaired equipment to the unit from which it came. Let unitD be one of the defenders. The fraction of its losses of type j equipment attributable to the attackers' type i weapons is computed as

$$\alpha(i,j) = \frac{A(i,j) * \mathrm{ersatk}(i)}{\sum_{k=1}^{IGA} A(k,j) * \mathrm{ersatk}(k)} .$$

Let IGA = *nggwep*(sideA). Let j0 = 0 if sideD = 1, and let j0 = nequip(1) if sideD = 2. The fraction of unit unitD's lost type j equipment that enters Repair Class p ($1 \le p \le 2$) is defined to be

$$\sum_{i=1}^{IGA} \mathit{frdmg}(p,i,j0+j) * (\alpha(i,j) * \mathrm{loss}(\mathrm{unitD},j)).$$

This quantity is added to any of the unit's type j equipment already in the repair class. The unit's current resources, [resources](unitD,j) is not increased; that will be done when the equipment is repaired. Now let unit unitA be one of the attackers. Let IGD = *nggwep*(sideD). For $1 \le i \le$ IGD and $1 \le j \le$ nequip(sideA), let

$$\alpha(i,j) = \frac{D(i,j) * \mathrm{ersdef}(i)}{\sum_{k=1}^{IGD} D(k,j) * \mathrm{ersdef}(k)} .$$

Let j0 = 0 if sideA = 1, and let j0 = nequip(1) if sideA = 2. The fraction of unitA's lost type j equipment that enters Repair Class p ($1 \le p \le 2$) is defined to be

$$\sum_{i=1}^{IGD} frdmg(p,i,j0+j) * (\alpha(i,j) * loss(unitA,j)).$$

6.2 FEBA MOVEMENT

Recall that each engagement has its own FEBA, measured by the variable feba, whose primary purpose is to determine when the attackers are allowed to occupy the engagement location. This subsection explains how any given engagement's feba is updated at the end of a frame to reflect the combat during the frame. The notation of Section 6.1 remains in force.

The change in feba from the start of the frame to the end of the frame depends on the attackers' posture, the defenders' posture, and a force ratio that includes the contribution of close air support (CAS). Air support is assessed at the start of every period, as Section 7 explains. (A period consists of one or more frames.) The losses of ground-to-ground weapons inflicted by CAS are recorded for use by the combat procedure. For every $1 \le j \le$ *nggwep*(sideD), let CASATK(j) be the amount of the defenders' type j weapons destroyed by air strikes made (by side sideA) in close support of the attackers; of course, if the attackers received no CAS in the period, CASATK(j) = 0. For every $1 \le j \le$ *nggwep*(sideA), let CASDEF(j) be the amount of the attackers' type i weapons destroyed by air strikes made (by side sideD) in close support of the defenders. These losses were determined at the start of the current period, and are assumed to be spread uniformly over the period. Therefore, to find CAS's effect on the engagement in the frame now ending, CASATK and CASDEF must be divided by nframe, defined as the number of frames in a period.

To find a force ratio that reflects both the ground forces and the air forces in the engagement, it is necessary to assign a value to CAS's contribution in a way consistent with the way the ground values are determined. The antipotential potential method facilitates this. Recall that the attackers' ground value is

$$fgrd = \sum_{i=1}^{m} ersatk(i) * vala(i),$$

where m = *nggwep*(sideA) (vala(i) = 0 if i > m). For every $1 \le i \le m$

$$\text{vala}(i) = (1/\text{alpha}) * \sum_{j=1}^{n} A(i,j) * \text{vald}(j),$$

where n = *nggwep*(sideD). Therefore,

$$\text{fgrd} = (1/\text{alpha}) * \sum_{j=1}^{n} \left(\sum_{i=1}^{m} A(i,j) * \text{ersatk}(i) \right) * \text{vald}(j).$$

The sum in parentheses is side sideD's total potential loss of type j materiel in the frame. The air value of side sideA in the engagement, fair, is computed the same way:

$$\text{fair} = (1/\text{alpha}) * \sum_{j=1}^{n} (\text{CASATK}(j) / \text{nframe}) * \text{vald}(j).$$

Analogously, the air value of side sideD in the engagement is defined as

$$\text{gair} = (1/\text{delta}) * \sum_{j=1}^{m} (\text{CASDEF}(j) / \text{nframe}) * \text{vala}(j).$$

The combined ground-air force ratio is

$$\text{FRGA} = \frac{\text{fgrd} + \text{fair}}{\text{ggrd} + \text{gair}} .$$

Let postA be the attackers' representative posture and postD the defenders' representative posture; postA and postD are defined in Section 6.1.3. The function value

vfeba (FRGA, postA, postD, sideA)

is the velocity of an engagement's FEBA when the combined ground-air force ratio is FRGA, the attackers' representative posture is postA, the defenders' representative posture is postD, and the attackers belong to side sideA. (The function vfeba is explicated in Section 6.4.) Let

$$\text{temp} = \text{vfeba}(\text{FRGA},\text{postA},\text{postD},\text{sideA}) * \textit{tframe}.$$

This number may be negative. If feba0 is the value of feba at the start of the frame, then at the end of the frame

$$\text{feba} = \min \{\max \{\text{feba0} + \text{temp}/\textit{depth},\ 0\},\ 1\},$$

6.3 BREAKPOINTS

6.3.1 Elimination

After the losses in one frame of an engagement are assessed, each of its attackers and defenders is examined to see if it is so weak it should be eliminated. The evaluation is based on its resources' "standard values": for s = 1 or s = 2 and $1 \leq i \leq nrs(s)$, rsvald(i,s) is the "standard value of a side s type i resource on defense". It is found by putting the resource on defense in a nominal engagement between a standard side s force and a standard enemy force. "Standard force" should not be confused with "standard target". By definition, the quantity of side s type j resources in a side s standard force is *stdfor*(s,j). Normally, but not necessarily, *stdtgt*(j,3-s) = *stdfor*(s,j) for every $1 \leq j \leq nrs(s)$. Let s1 = 1 and s2 = 2. Let nm = nmat(s2). For every $1 \leq i \leq$ *nggwep*(s1) and $1 \leq j \leq$ *nggwep*(s2), let

$$\mathrm{DSTD}(i,j) = \mathrm{kdef}(i,j,s1) * \mathit{aggdef}(i,j,s1) * \frac{\mathit{stdfor}(s2,j)}{\mathit{stdtgt}(j,s1)} * \frac{1}{\mathrm{DEN}}$$

where

$$\mathrm{DEN} = \sum_{k=1}^{nm} \mathit{aggdef}(i,k,s\ ) * \frac{\mathit{stdfor}(s2,k)}{\mathit{stdtgt}(k,s1)}.$$

Now let nm = nmat(s1), and for every $1 \leq i \leq$ *nggwep*(s2) and $1 \leq j \leq$ *nggwep*(s1), let

$$\mathrm{ASTD}(i,j) = \mathrm{katk}(i,j,s2) * \mathit{aggatk}(i,j,s2) * \frac{\mathit{stdfor}(s1,j)}{\mathit{stdtgt}(j,s2)} * \frac{1}{\mathrm{DEN}},$$

where

$$\mathrm{DEN} = \sum_{k=1}^{nm} \mathit{aggatk}(i,j,s2) * \frac{\mathit{stdfor}(s1,k)}{\mathit{stdtgt}(k,s2)}.$$

The subprogram app is called with the kill matrices ASTD and DSTD as arguments; it returns the values of the side s1 resources (on defense), which define rsvald(*,s1). The values of the side s2 resources (on attack) define rsvala(*,s2)--the "standard values of side s2 resources on attack"--which are used to resolve mutual attacks (Section 3.4.4). To compute rsvald(*,s2)--the Blue resources' standard values on defense--the process is repeated with s1 = 2 and s2 = 1.

Let unit ibu be an attacker or defender in the engagement. Let s = 1 if it belongs to Red and s = 2 if it belongs to Blue. Let n = nrs(s). Let

$$sv = \sum_{i=1}^{n} \mathit{toe}(\mathit{butype}(\text{ibu}),i) * \text{rsvald}(i,s),$$

$$cv = \sum_{i=1}^{n} [\text{resources}](\text{ibu},i) * \text{rsvald}(i,s).$$

If

$$cv < \mathit{vanish}(\mathit{butype}(\text{ibu})) * sv - 10^{-5},$$

then unit ibu is eliminated: it is assigned the mission whose only order declares -10 as the desired posture.

6.3.2 Retreat of Defenders

If, at the end of a frame, feba ≥ *febad* in a given engagement, the defenders are declared defeated, and the combat procedure calls the tactical subprogram haven to ascertain whether the defenders have a line of retreat. To be admissible as a direction of retreat, a cell must be active and adjacent to the engagement location, and it must satisfy the following conditions: (i) it contains none of the attackers in the engagement; (ii) if it contains an active unit belonging to the attackers' side, then it must also contain an active unit belonging to the defenders' side and be owned by the defenders' side. If the game design variable *haven.zoc* has the value .true., a direction of retreat can also be blocked by the presence of attackers in cells flanking it. To be precise, suppose cell i satisfies all the preceding conditions for admissibility as a direction of retreat; if *haven.zoc* = .true., cell i must satisfy the additional condition: (iii) if one of the attackers is located in a cell adjacent to cell i, then cell i must contain an active unit from the defenders' side and must be owned by the defenders' side.

If the defenders have no admissible direction of retreat, they are eliminated. If they have an admissible direction of retreat, haven selects the most desirable one. Each admissible direction of retreat is scored as follows:

(1) Initially, let its score be 0.

(2) If it is exactly two cells away from a cell containing one of the attackers, let its score be -1.

(3) If it is adjacent to a cell (other than the engagement location) containing one of the attackers, let its score be -2.

(4) If it is owned by the attackers' side, decrease its score by .5.

(5) If it is owned by the defenders' side and contains an active unit from their side, increase its score by 1.8.

(6) Let s = 1 if the defenders are Red and s = 2 if they are Blue. If the cell is the k-th rim cell of the engagement location, increase its score by .01 * *haven.love*(s,k).

The "rim cells" of a given cell are the cells adjacent to it. They are ordered by number, from lowest to highest. For example, in Figure 3.3 (page 3-6), the first rim cell of cell 6 is cell 2, the second is cell 3, the third is cell 5, the fourth is cell 7, the fifth is cell 9, and the sixth is cell 10; the fourth rim cell of cell 1 is cell 2, the sixth is cell 5, and the other rim cells of cell 1 do not exist; the fifth rim cell of cell 14 is cell 17.

The game design datum *haven.love*(s,k) measures the intrinsic desirability to a side s unit of retreating toward the k-th rim cell; ordinarily, *haven.love*(s,k) should be greatest for k such that the k-th rim cell leads in the direction of the side s COMMZ and least for k such that the k-th rim cell leads in the direction of the enemy COMMZ. To prevent *haven.love* from dominating other considerations in selection of a direction of retreat, it should be fairly small; $0 \leq$ *haven.love*(s,k) ≤ 6 is recommended.

Let cell r be the admissible direction of retreat with the highest score; ties are broken by minimizing r. Each defender that is not already disengaging is forced to disengage immediately toward cell r: it is assigned the active order

desired objective = r,
desired posture = = *pmapup*(*pmapup*(*pmapup*(*pmapup*(p)))),

where p is its current posture (a hold posture). Once all the defenders are disengaging, the attackers are allowed to occupy the engagement location, as Section 3.4.3 explains.

6.3.3 Withdrawal of Attackers

The engagement attackers automatically quit trying to seize the engagement location if the combined ground-air force ratio, FRGA, becomes too low. As before, let side sideA be the attackers' side, and let posture postA be their representative

posture. If the attackers' location is not the engagement location and

$$\text{FRGA} < \mathit{quit}(\text{postA},\text{sideA}),$$

then each attacker is assigned a new mission ordering it into posture *pmapup*(pp), where posture pp is its present posture, at its present location; execution of the mission terminates the attack and therefore the engagement.

6.4 THE COMBAT FUNCTIONS

This subsection explains the functions frinv, zrarea, freff, prep, frdval, and vfeba, which the subprogram combat invokes. They are based on piecewise-affine (loosely speaking, piecewise-linear) functions mapping the real line into the real line. Such a function is specified by listing points in its domain--x(1), x(2),...,x(n)--and its value at each of these points--y(1), y(2),...,y(n). By requirement, $x(1) \le x(2) \le \ldots \le x(n)$. The function pafgen evaluates a piecewise-affine function given these points. Let w be a real number. If $w \le x(1)$, then

$$\text{pafgen}(w,y,x) = y(1).$$

If $w \ge x(n)$, then

$$\text{pafgen}(w,y,x) = y(n).$$

Suppose $x(1) < w < x(n)$. Let

$$i1 = \max\,\{i:\ \ x(i) \le w,\ 1 \le i \le n\},$$

$$i2 = \min\,\{i:\ \ x(i) > w,\ 1 \le i \le n\}.$$

Then

$$\text{pafgen}(w,y,x) =$$

$$y(i1) + \frac{w - x(i1)}{x(i2) - x(i1)} * (y(i2) - y(i1)).$$

(The IDAHEX function pafgen actually has an additional argument--n, the number of components of the vector x or y.)

A similar function, paf, is used to evaluate a piecewise-affine function whose domain is the nonnegative reals. Such a function is specified by listing its value at 0, which is denoted y0; listing points in its domain--x(1),...,x(n); and listing its values at these points--y(1),...,y(n). By requirement, $0 \le x(1) \le \ldots \le x(n)$. Let w be a real number. Define the vector ylong, with n+1 components, as follows:

$$\text{ylong}(1) = \text{y0},$$

$$\text{ylong}(i+1) = y(i) \text{ for } 1 \leq i \leq n.$$

Define the vector xlong, with n+1 components, as follows:

$$\text{xlong}(1) = \text{x0},$$

$$\text{xlong}(i+1) = x(i) \text{ for } 1 \leq i \leq n.$$

Then

$$\text{paf}(w,y0,y,x) = \text{pafgen}(w,\text{ylong},\text{xlong}).$$

(The IDAHEX function paf actually has an additional argument--n, the number of components of the vector x.)

6 4.1 Resource Availability for Combat - frinv

The function frinv, as called by the combat procedure, has two essential arguments: a unit number, ibu; and a resource type, irsarg. If the unit's resources of type irsarg are equipment (weapons or transport), frinv returns the fraction of them that are available for combat; equipment is available for combat if and only if its requirements for support and protection are met. If the type irsarg resources are support resources (supplies or personnel), frinv returns the fraction of them that are available and required. The neutral term "fractional involvement" designates the number returned in either case. In the process of determining the fractional involvement of type irsarg resources, frinv determines the fractional involvement of every type of resources in unit ibu. Let s = 1 if unit ibu is Red and s = 2 if it is Blue. Let fi(irs) be the fractional involvement of type irs resources in unit ibu for every $1 \leq \text{irs} \leq \text{nrs}(s)$. The sequel explains how it is determined.

Given that frinv has been called by the combat procedure, unit ibu must be engaged. Other units from its side may be participating in the same engagement; frinv assumes that all such units with the same location as unit ibu perform as an integral whole, sharing their support and using their weapons in concert. Let L be the set of every unit that is from the same side, participating in the same engagement, and located in the same cell as unit ibu. For $1 \leq \text{irs} \leq \text{nrs}(s)$, define amount irs as the total quantity of type irs resources held by the force L:

$$\text{amount}(\text{irs}) = \sum_{i \in L} [\text{resources}](i,\text{irs}).$$

Let

$$nsp = \mathit{nss}(s) + \mathit{npers}(s),$$

the number of types of side s support resources. Suppose nsp = 0. Then fi is determined solely by considerations of equipment protection. The game design variable *pg* organizes equipment into protection groups, numbered 1, 2,...; *pg*(i,s) is the protection group to which type i equipment of side s belongs. At least one type of the side's ground-to-ground weapons must belong to protection group 1. Any equipment in protection group 1 can protect itself and equipment in higher protection groups. Equipment in a protection group higher than 1 cannot protect itself, but can protect equipment in protection groups higher than its own. The quantity of type i equipment that a unit-quantity of type j equipment can protect, provided $pg(j) < pg(i)$, is *prot*(i,j,s) by definition. Equipment other than ground-to-ground weapons, although it may conceivably belong to protection group 1 and be able to protect itself, is assumed to be unable to protect other equipment--i.e., *prot*(i,j,s) is assumed to be 0 if $j > \mathit{nggwep}(s)$. In the present case, where support is ignored, fi(i) = 1 for every i such that *pg*(i,s) = 1. The fractional involvement of equipment in higher protection groups, if any, is determined inductively. Suppose that for some

$$k < \max \{pg(i,s);\ 1 \le i \le \mathrm{nequip}(s)\}$$

fi(i) has been determined for every i in the set

$$I = \{i:\ pg(i,s) \le k,\ \ 1 \le i \le \mathrm{nequip}(s)\}.$$

For each j such that *pg*(j,s) = k + 1, let

$$QP(j) = \sum_{i \in I} \mathit{prot}(j,i,s) * fi(i) * \mathrm{amount}(i),$$

the quantity of type j equipment that can be protected. Set

$$fi(j) = \frac{\min \{QP(j),\ \mathrm{amount}(j)\}}{\mathrm{amount}(j)}.$$

That completes the induction step. If possible, k is incremented by 1 and the step is repeated.

Typically, small arms belong to protection group 1, tanks to group 2, artillery to group 3, and ground-to-air weapons and transport to group 4. Notice that protecting one type of equipment does not reduce a weapon's ability to protect other types of equipment. One might think of the equipment in protection group 1 as being deployed near the front of the formation, the equipment in protection group 2 deployed behind it, and so on,

with the equipment in each protection group acting as a screen for the equipment deployed behind it.

Henceforth, assume that nsp > 0. To determine fi(i) for every $1 \leq i \leq nrs(s)$, frinv implicitly allocates support to the various types of resources. The allocation is reasonable, but not optimal: it does not maximize the force L's value in combat. It should not; the allocation is partly prescriptive. It is designed to field a balanced combat force--one in line with *stdfor*(s,*), with no unprotected equipment.

Let

$$neq = nequip(s).$$

For every $1 \leq k \leq nsp$ let

$$suppt(k) = amount(neq+k),$$

the total quantity of type k support held by the units in L. If personnel are played--i.e., if *npers*(s) > 0--the quantities of personnel available to support materiel must be reduced by overhead requirements:

$$suppt(k) \longleftarrow suppt(k) - \sum_{i \in L} \mathit{ppoh}(k, \mathit{butype}(i))$$

for every $1 \leq k \leq \mathit{npers}(s)$. (*ppoh*(k,j) is defined as the overhead of type k personnel in a type j battle unit--a quantity that is independent of the unit's actual size.)

Let

$$i0 = \begin{cases} 0 & \text{if } s = 1, \\ nrs(1) & \text{if } s = 2. \end{cases}$$

The game design datum *spdd*(k,i0+irs) is the demand of a unit-quantity of side s type irs resources ($1 \leq irs \leq nrs(s)$) for type k support ($1 \leq k \leq nsp$).[1] The IDAHEX computer program assumes that supplies' demand for supplies is 0 and personnel's demand for personnel is 0. The total demand of the force's resources of type irs for support of type k is computed as

---

[1]Equipment's requirement for support normally should include the personnel needed to operate it in combat and, in addition, personnel needed to keep it operational (by maintenance and repair, for example). The game designer must avoid counting personnel requirements twice--once in resources' requirements (*spdd*) and once in overhead (*ppoh*).

$$dd(k) = amount(irs) * \mathit{spdd}(k,i0+irs).$$

Suupose irs $\le$ nmat(s). Let ss(k) be the quantity of type k support allocated to the force's type irs materiel. For every $1 \le k \le nsp$, let

$$sigma(k) = paf\ (ss(k)/dd(k),\ \mathit{frinv.f0}(k,i0+irs),\ \mathit{frinv.f}(k,i0+irs,*),\ \mathit{frinv.x}(s,*))$$

if dd(k) > 0, and let sigma(k) = 1 if not. The fractional involvement of the force's type irs materiel, fi(irs), is given by

$$fi(irs) = \min \{sigma(k);\ 1 \le k \le nsp\}.$$

Thus, allocation of support to each type of materiel determines its fractional involvement. To be sure that no more of any type of support is allocated than is available, the vector alloc is used to keep track of the allocation; alloc(k), for $1 \le k \le nsp$, is the total quantity of type k support allocated. Initially, alloc $\equiv$ 0.

First, personnel are allocated to supplies. For each $1 \le kpp \le \mathit{npers}(s)$, the total demand for type kpp personnel by the force's supplies is

$$Q = \sum_{kss=1}^{\mathit{nss}(s)} suppt(kss) * \mathit{spdd}(\ \mathit{nss}(s)+kpp,\ i0+neq+kss);$$

the demand of type kss supplies alone is

$$dd = suppt(kss) * \mathit{spdd}(\ \mathit{nss}(s)+kpp,\ i0+neq+kss)$$

($1 \le kss \le \mathit{nss}(s)$); the allocation of type kpp personnel to type kss supplies is chosen as

$$\min \{dd,\ dd * (suppt(kpp)\ /\ Q)\},$$

and alloc(nss(s)+kpp) is increased by this quantity. As explained above, the allocation determines fi(kss). Only that fraction of type kss supplies are available for allocation; redefine suppt(kss) for every $1 \le kss \le \mathit{nss}(s)$:

$$suppt(kss) \leftarrow fi(kss) * suppt(kss).$$

Next, supplies are allocated to personnel, but fi(irs) is set to 1 for each nmat(s) < irs $\le$ nrs(s) whether or not the allocation satisfies personnel's demand. Record the allocation of supplies:

$$alloc(kss) \leftarrow alloc(kss) + \sum_{kpp=1}^{npers(s)} amount(nmat(s)+kpp) * spdd(kss, i0+nmat(s)+kpp).$$

(If *nss*(s) = 0 or *npers*(s) = 0, both preceding steps are vacuous.)

Next, support is allocated to equipment. Let

$$I = \{ieq: \ stdfor(s,ieq) > 0, \ 1 \le ieq \le neq\}.$$

If $i \le neq$ but $i \notin I$, fi(i) is set to 0 and never changed. If $i \in I$ and amount(i) = 0, fi(i) is set to 1 and never changed. Initially, fi(i) = 0 for every other $i \in I$. It is increased in small increments by increasing the support allocated to each type of equipment in the set I. Let rgain be a small positive number--.01, for example. At the start of any given iteration of the algorithm, let

$$q(j) = fi(j) * amount(j)$$

for every j. (fi may have been redefined in prior iterations.) The iteration consists of performing the following sequence of operations for each $i \in I$ for which amount(i) > 0.

Step 1: If *pg*(i,s) = 1, let qp = $+\infty$ and go to Step 2. Let P be the set of every $j \in I$ such that *pg*(j,s) < *pg*(i,s). Let

$$qp = \sum_{j \in P} prot(i,j,s) * q(j),$$

the quantity of type i equipment that can be protected by the equipment presently available for combat.

Step 2: Let

$$qr = rgain * stdfor(s,i),$$

the amount by which q(i) would have to increase in order to increase

$$q(i) \ / \ stdfor(s,i)$$

by the amount rgain. Let

$$qadd = \min \{qp, qr\}.$$

For every $1 \le ksp \le nsp$, let REQ(ksp) be the amount of additional type ksp support that must be allocated to

type i equipment to increase q(i) by the amount qadd--i.e., to increase fi(i) by the amount

$$\text{qadd} \;/\; \text{amount(i)}.$$

If

$$\text{alloc(ksp)} + \text{REQ(ksp)} \le \text{suppt(ksp)}$$

for every $1 \le \text{ksp} \le \text{nsp}$, then allocate the support and update fi and q(i):

$$\text{alloc(ksp)} \longleftarrow \text{alloc(ksp)} + \text{REQ(ksp)} \text{ for every } 1 \le \text{ksp} \le \text{nsp},$$

$$\text{fi(i)} \longleftarrow \text{fi(i)} + \text{qadd} \;/\; \text{amount(i)},$$

$$\text{q(i)} \longleftarrow \text{fi(i)} * \text{amount(i)}.$$

If not, fi(i) cannot be increased.

Thus, the algorithm tends to field a balanced combat force--i.e., it strives to equalize

$$\frac{\text{fi(i)} * \text{amount(i)}}{\textit{stdfor}\text{(s,i)}}$$

over every $i \in I$ and never commits unprotected equipment to combat.

The fractional involvement of support resources as computed above may be too high since support resources should be involved in combat (and subject to enemy fire) only to the extent that they are actually in demand. Therefore, fi(ksp) is redefined as follows for each $1 \le \text{ksp} \le \text{nsp}$:

$$\text{fi(ksp)} \leftarrow \min \{\text{fi(ksp)}, \text{demand} \;/\; \text{amount(neq+ksp)}\},$$

where

$$\text{demand} = \sum_{\text{irs}=1}^{\text{nrs(s)}} \textit{spdd}\text{(ksp,i0+irs)} * \text{amount(irs)}.$$

The preceding explains the derivation of frinv(irsarg,ibu) in the case where unit ibu is engaged; that case always applies when frinv is called by the combat procedure. The derivation actually involves finding the fractional involvement of resources in a set of units, L, that share their support and use their weapons in concert. Sometimes, for a player's information, it is useful to find the fractional involvement for a specified force L, rather than a force inferred by frinv from the argument ibu. The IDAHEX entry point frinv actually has two additional

arguments: a vector named list and an integer variable named nlist. If ibu $\leq$ 0, frinv constructs the set L from the vector list, whose first nlist elements must be the identification numbers of battle units all from the same side; frinv then proceeds as above to find the fractional involvement of resources.

6.4.2 Area of Area of Influence - zrarea

The function value zrarea(ibu) is the area of the area of influence of battle unit ibu. It is 0 if the unit is inactive. Assume unit ibu is active. Let s = 1 if it is Red and s = 2 if it is Blue. Its current value, measured in terms of the standard resource values, is

$$cv = \sum_{irs=1}^{nrs(s)} \text{rsvald}(irs,s) * [\text{resources}](ibu,irs).$$

Its value at *toe* strength would be

$$sv = \sum_{irs=1}^{nrs(s)} \text{rsvald}(irs,s) * \textit{toe}(\textit{butype}(ibu),irs).$$

The size of its area of influence is assumed to be proportional to the size at *toe* strength. The latter depends upon the unit's type and posture class. Let pc be the unit's posture class:

$$\text{zrarea}(ibu) = (cv/sv) * \textit{aisize}(\textit{butype}(ibu),pc).$$

6.4.3 Battle Unit Effectiveness - freff

A battle unit's effectiveness in combat may depend upon the density of friendly forces in its location. If the density is too low, the friendly force is vulnerable to infiltration and turning more vulnerable to area fire, and congestion of the trafficable areas reduces the maneuver battalions' tactical mobility. In many models the degradation of effectiveness due to high density is implemented indirectly by a rule limiting the number of units located in the same cell or sector. Since units may vary greatly in size, especially late in the game, IDAHEX uses a more flexible method.

Suppose the location of unit ibu, an active battle unit, is cell i. Let s = 1 if the unit is Red and s = 2 if Blue. Let F be the set of every active side s unit in cell i. The total area of their areas of influence is

$$A = \sum_{j \in F} \text{zrarea}(j).$$

The friendly force density, d, equals A divided by the cell area.

$$y = \text{paf}(d,\ \textit{freff.f}0(s),\ \textit{freff.f}(s,*),\ \textit{freff.x}(s,*)).$$

Suppose unit ibu is in an attack posture. Its fractional effectiveness is given by

$$\text{freff(ibu)} = \begin{cases} 1; & d \le 1 \\ y; & d > 1 \end{cases}.$$

Alternatively, suppose unit ibu is in a hold or disengagement posture. Let G be the set of every active side s unit in a hold or disengagement posture in cell i. Let

$$B = \sum_{j \in G} \text{zrarea}(j),$$

and let dd equal B divided by the cell area. Let

$$z = \text{paf}(dd,\ \textit{freff.f}0(s),\ \textit{freff.f}(s,*),\ \textit{freff.x}(s,*)).$$

The fractional effectiveness of unit ibu is given by

$$\text{freff(ibu)} = \begin{cases} y & ;\ dd \ge 1 \\ \min\{y,z\}; & dd < 1 \end{cases}.$$

6.4.4 Defensive Preparation - prep

The vulnerability of materiel varies with the time its battle unit has had to prepare a defense. Suppose s = 1 or s = 2, and suppose $1 \le i \le \text{nmat}(s)$. The function value prep(i,s,h) is the factor the combat procedure applies to type i materiel belonging to a side s unit whose defense preparation time equals h.

$$\text{prep}(i,s,h) = \text{pafgen}\ (h,\ \textit{prep.f}(i,s,*),\ \textit{prep.x}(s,*)).$$

Because of peculiarities in the way preparation time is calculated, h may be negative. The game designer should allow for

this possibility by choosing

$$prep.x(s,1) > 0.$$

6.4.5 Close Supporting Fire - csf

Units that are not engaged may fire in (close) support of friendly units in an engagement. The supporting units need not be in or adjacent to the engagement location, but their distance from the engagement location will affect the amount of fire they can contribute. Close supporting fire is directed against all the enemy units in an engagement; units can not be targeted specifically, and unengaged units can never be targeted. To contribute supporting fire, a unit must be in a hold posture and unengaged.

The function csf has three arguments; element variables ngn and s, and a vector esf. The function finds the effective quantities of supporting fire for side s (s=1 or s=2) in engagement ngn and returns them in the vector esf.

Close supporting fire is a "special activity" requested by the players. (See Volume 3, Section 4.) Other special activities affect the lines of communication; they are discussed in Section 9.2. Special activities are recorded in the Special Activities List, whose columns are the vectors saloc1, saloc2, satype, saunit (which is not used for supporting fire), and salvl; saloc1(k) = 0 signifies that line k of the list is unused. If saloc1(k) > 0, satype(k) indicates the type of special activity specified by line k of the List; satype(k) = 10 signifies close supporting fire. Close supporting fire by side s (s = 1 or s = 2) is permitted and game design data concerning it are expected if and only if the game design datum *ifsa*(10,s) has the value true.

Suppose, for some ksa ≥ 1, saloc1(ksa) > 0 and satype(ksa) = 10. Let

orig = saloc1(ksa).

The item orig is a cell number. Suppose that [owner](orig) = s. Then special activity ksa is a close supporting fire activity for side s. The fire comes from side s units in cell orig. The fire goes to cell saloc2(ksa); to be precise, the fire contributes to an engagement in the cell. Suppose that cell saloc2(ksa) is the location of engagement ngn. If engagement ngn is the only engagement in the cell, special activity ksa is close supporting fire for side s in engagement ngn, and is relevant to finding esf. (Two engagements can arise in the same cell if a task force attacks a cell from which friendly

units are disengaging.) Alternatively, if there is another engagement in cell saloc2(ksa), engagement ngn gets the supporting fire if and only if side s is the defender in engagement ngn. Assume that either condition holds, so that special activity ksa represents close supporting fire for side s in engagement ngn. Let

$$h = \mathrm{salvl(ksa)}.$$

It is a player-selected limit on the intensity of fire ($0 < h \leq 1$).

Let S be the index set of every unengaged side s unit in a hold posture in cell orig; S is the set of firing units. The goal is to find the quantity of supporting fire each type of their weapons contributes to engagement ngn--actually, to find the effective number of weapons firing. The game design array *rngfc* reduces the effective number firing according to the range to the target. (Remember that a given type of weapon may embrace a variety of weapons with different ranges.) The datum *rngfc* (i,s,k) is the fraction of the type i weapons within range if the range to the engagement location is *range*(s,k). For a range intermediate between two points, *range*(s,k) and *range*(s,k+1), the fraction of weapons capable of effective fire is found by interpolation (using the function pafgen, which is explained in Section 6.4). The game design array *postfc* reduces the number of weapons firing according to the units' hold postures. The array *spregf* is used to limit the number firing due to shortages of supplies or personnel.

Step 1. Let nw = *nggwep*(s). Let d be the straight-line distance from the center of cell orig to the center of cell saloc2(ksa). For $1 \leq i \leq nw$, let

$$f(i) = \mathrm{pafgen}(d,\ \mathit{rngfc}(i,s,*),\ \mathit{range}(s,*)).$$

Step 2. For $1 \leq i \leq nw$, let

$$q(i) = \sum_{k \in S} h * f(i) * \mathit{postfc}(i,s,\mathrm{bupost}(k)-9) * [\mathrm{resources}](k,i).$$

(Recall that bupost(k) is the posture of unit k and that every unit in s is in a hold posture.)

Step 3. Let n = nsp(s), the number of types of side s support resources. If n = 0, skip this step. The total demand for type j support resources is determined to be

$$dd(j) = \sum_{i=1}^{nw} \mathit{spreqf}(j,i,s) * q(i).$$

Let ss(j) be the total quantity of type j support resources held by the units in s. Let

$$\text{lambda} = \min\{ss(j)/dd(j);\ 1 \le j \le n\},$$

where 0/0 is defined to be $+\infty$. If lambda > 1, set lambda = 1. Redefine q(i) for $1 \le i \le nw$ to reflect support constraints:

$$q(i) = \text{lambda} * q(i).$$

Step 4. Assess the additional consumption of supplies caused by the supporting fire activity: for every $k \in L$ and every $1 \le j \le \mathit{nss}(s)$,

$$[\text{resources}](k,nwep(s)+j) \leftarrow [\text{resources}](k,nwep(s)+j) - \text{lambda} * dd(j).$$

The preceding steps determine the effective number of type i weapons ($1 \le i \le \mathit{nggwep}(s)$) contributing supporting fire to side s in engagement ngn as a result of special activity ksa. Finding the total effective number for all such special activities yields the vector esf.

6.4.6 Fraction of Value Lost - frdval

This function finds the fraction of value that a side in an engagement loses given the side's posture and the engagement's ground force ratio. Let post be the side's posture and FR the force ratio. Let k = *poff*(post). Let

$$\text{temp} = \text{paf}\,(FR,\ \mathit{frdval.f0atk}(k),\ \mathit{frdval.fatk}(k,*),\ \mathit{frdval.x})$$

if post $\ge$ 40 (the side is the attacker in the engagement), and

$$\text{temp} = \text{paf}\,(FR,\ \mathit{frdval.f0def}(k),\ \mathit{frdval.fdef}(k,*),\ \mathit{frdval.x})$$

if post < 40 (the side is the defender). The number temp gives the fraction of value lost in one unit of time, but the combat procedure needs to know the fraction lost in one frame. Therefore,

$$\text{frdval (FR, post)} = \begin{cases} 1 - (1 - \text{temp})^{**}\text{tframe}; & \text{temp} \geq 0 \\ \text{temp}; & \text{temp} < 0. \end{cases}$$

The combat procedure, which calls frdval, interprets frdval(FR,post) < 0 as a signal that no prediction of the side's losses should be made from the force ratio and therefore that the side's losses should not be scaled according to it.

6.4.7 FEBA Velocity - vfeba

The function value vfeba (FR, pa, pd, sa) is the velocity of the FEBA (measured by *depth* * feba) in an engagement in which the force ratio is FR, the attackers are from side sa, the attackers are in posture pa, and the defenders are in posture pd. Let

$$\text{ka} = \begin{cases} \mathit{poff}(\text{pa}) & ; \quad \text{sa} = 1 \\ \mathit{poff}(\text{pa}) + \text{vfeba.npa}; & \text{sa} = 2. \end{cases}$$

The offset vfeba.npa is defined by the entry point vfeba0. If the number of attack postures is large (i.e., if *npost*(4) is close to 10), it may be necessary to increase vfeba.npa and the dimensions of certain variables declared by vfeba0. In that event, IDAHEX will advise the game designer with a message in file 51. Let kd = *poff*(pd). Then

$$\text{vfeba (FR, pa, pd, sa)} = \text{paf (FR, } \mathit{vfeba.f0}\text{(ka,kd), } \mathit{vfeba.f}\text{(ka,kd,*), } \mathit{vfeba.fr}\text{).}$$

This number may be negative.

In defining *vfeba.f0* and *vfeba.f* the game designer should keep in mind that the attackers have already been charged with the time needed to go from their location to the engagement location, and if they occupy the engagement location and then leave, they will be charged with the time needed to go from the engagement location to their new locations; the movement delay takes care of unopposed movement. The feba velocity is used to determine an *additional* delay caused by opposition. Consequently, if the force ratio is very high, the feba velocity should be very high; it should not be limited by the unopposed movement rate.

7. AIR SUPPORT

At the start of every period (including t = *tinit*), each player may enter air strikes. IDAHEX contains no air warfare model and therefore has no way of ascertaining what air assets a side can allocate against enemy ground forces. It assumes that any air strike a player enters is within his side's capability. In practice, the game designer adopts either of two solutions: he gives each player a list of the air assets available in each cycle for use against enemy ground forces, or he runs an air warfare model concurrently with IDAHEX. The first solution is suitable when the course of the air war is easy to predict--usually because one side clearly dominates.

Suppose the side s player (s=1 or s=2) is inputting an air strike. He must choose a "strike role" from the following list:

role number	role name
1	barrier intensification
2	railroad damage
3	road damage
4	deep air support
5	close air support (CAS)

Barrier intensification typically represents an attack on bridges. The first three strike roles all imply an effort to degrade the line of communication between two adjacent cells. These "LOC modification activities" are explicated in Section 9. This section concentrates on strike roles 4 and 5. Close air support is an attack on enemy battle units participating in a specific engagement with friendly battle units. Deep air support is any other attack on enemy battle units.

After specifying the strike role, the player must tell IDAHEX where the strike should occur. This means specifying the engagement in which close air support is desired if the strike role is 5, or the "target cell" if the strike role is 4, or the "target LOC" if the strike role is 1, 2, 3; the player identifies a target LOC by identifying the two adjacent cells it links.

If the strike role is deep air support the player must define the four-component vector asprty, which is a list of the four positive posture classes in order of priority. It and the target cell together determine which battle units come under attack.

Finally, the player must set ascomp; ascomp(i) is the number of type i aircraft participating in the strike ($1 \le i \le nactyp(s)$).

Suppose the air strike role is CAS. Let V be the set of every enemy unit in the engagement, identified by unit number. If the enemy units are the defenders in the engagement, and if at least one of them is in a hold posture, then delete from V every unit in a disengagement posture.

On the other hand, suppose the strike role is support. Let k be the smallest integer such that asprty(k) equals the posture class of some active enemy unit located in the target cell. Thus, asprty(k) is the highest-priority posture class that appears among enemy units in the target cell. Let pc = asptry(k). Define V as the set of every active enemy unit, identified by unit number, whose location is the target cell and posture class is pc. These units are the targets of the strike. This definition of V implicitly assumes that the strike aircraft can only distinguish enemy units from each other by location (cell) and posture class.

Let

$$\mathrm{nw} = nagwep(\mathrm{s}).$$

For every $1 \le iw \le nw$, the amount of type iw air-to-ground weapons in the strike is

$$\mathrm{agwep(iw)} = \sum_{i=1}^{n} agload(\mathrm{i,iw,s}) * \mathrm{ascomp(i)}$$

where $n = nactyp(s)$. Let $v = 3 - s$. (Side v is the enemy of side s.) For $1 \le j \le nmat(v)$, the amount of type j materiel in the target battle units is

$$\mathrm{grdrs(j)} = \sum_{i \in V} [\mathrm{resources}](\mathrm{i,j}).$$

Let env be the environment type of the target cell: if the target cell is cell i,

$$\mathrm{env} = [\mathrm{environment}](\mathrm{i}).$$

Choose an air-to-ground weapon type, iw; $1 \le iw \le nw$. For every $1 \le j \le nmat(v)$, define

$$aag(j) = \begin{cases} \mathit{aagatk}(iw,j,s) & \text{if the strike role is CAS and side s is the engagement attacker,} \\ \mathit{aagdef}(iw,j,s) & \text{if the strike role is CAS and side s is the engagement defender,} \\ \mathit{aagred}(iw,j,pc) & \text{if the strike role is deep air support and } s = 1 \\ \mathit{aagblu}(iw,j,pc) & \text{if the strike role is deep air support and } s = 2 \end{cases}$$

(Recall that pc is the posture class of the target battle units in the case where the strike role is general air support.) For $1 \le j \le nmat(j)$, the fraction of fire of type iw weapons allocated to the target units' type j materiel is computed as

$$alpha(j) = \frac{aag(j) * (grdrs(j) / \mathit{stdtgt}(j,v))}{\sum_i aag(i) * (grdrs(i) / \mathit{stdtgt}(i,v))} .$$

This method of allocating fire is analogous to the method used in ground combat.

Choose ibu ε V and $1 \le j \le nmat(v)$. The goal is to determine the potential destruction of type j materiel in unit ibu by the type iw weapons in the strike, denoted K(iw,j,ibu). In parallel with the ground combat attrition procedure, this quantity is found by taking a basic kill rate and applying factors that each adjust either the shooting weapon's lethality or the target materiel's vulnerability. The basic kill rate depends upon the game design datum *kag*(iw,j,s) and the allocation of fire. The adjustment factors depend upon the posture class of unit ibu--denoted by pc--and the target cell environment. By definition,

$$\begin{aligned} K(iw,j,ibu) = \mathit{kag}(iw,j,s) & * \mathit{fcagrp}(iw,s,pc) * \mathit{fcagcp}(j,v,pc) \\ & * \mathit{fcagre}(iw,s,env) * \mathit{fcagce}(j,v,env) \\ & * Q, \end{aligned}$$

where Q is the amount of fire from type iw weapons allocated to type j materiel in unit ibu:

$$Q = \left(alpha(j) * ([resources](ibu,j) / grdrs(j)) \right) * agwep(iw).$$

The total potential loss of type j materiel by all the target units is

$$\sum_{i \varepsilon V} \sum_{iw=1}^{nw} K(iw,j,i).$$

If the strike role is CAS and $j \leq$ *nggwep*(v), this quantity is recorded in the array casfx for later use by the combat procedure.

Choose ibu ε V and $1 \leq j \leq$ nmat(v). The actual loss of type j materiel by unit ibu is computed as follows. Initially, set iw = 1. Let

$$L = \min \{K(iw,j,ibu), [resources](ibu,j)\},$$

and reduce the unit's stocks of type j materiel by that quantity:

$$[resources](ibu,j) \longleftarrow [resources](ibu,j) - L.$$

This loss of type j materiel implies a loss of personnel. If unit ibu is Red, then for each $1 \leq k \leq$ *npers*(1), the number of type k personnel in the unit is reduced by the quantity

$$\min \{dgpred(k,iw,j) * L, [rescurces](ibu,nmat(1)+k)\}.$$

If unit ibu is Blue, then for each $1 \leq k \leq$ *npers*(2), the number of type k personnel in the unit is reduced by the quantity

$$\min \{dgpblu(k,iw,j) * L, [resources](ibu,nmat(2)+k)\}.$$

If iw < nw, iw is incremented by 1 and the process (starting with the definition of L) is repeated. The preceding is an efficient way of computing the attrition, but leads to an unfortunate anomaly: the way the air-to-ground weapons are ordered can affect personnel losses. The anomaly arises only when the battle unit has some type j materiel but so little that

$$\sum_{iw=1}^{nw} K(iw,j,ibu) > [resources](ibu,j)$$

(before [resources](ibu,j) is reduced). Losses of materiel are never affected by the ordering of air-to-ground weapons.

8. LOGISTICS

8.1 SUPPLIES CONSUMPTION

Every unit's consumption of supplies is assessed at the end of each frame, immediately after all engagements are evaluated and the resulting attrition is assessed. An inactive unit (one in posture class -1 or 0) consumes no supplies; this section applies only to active units.

Let s = 1 or s = 2. If *nss*(s) = 0--side s supplies are not represented--then nothing is done. Otherwise, consumption of supplies by side s battle units in a given frame is assessed as follows:

Let unit ibu be a side s battle unit. Let $1 \leq k \leq nss(s)$. Denote the unit's demand for type k supplies by D(ibu,k). Two cases arise:

Case 1: unit ibu is not engaged. If it is not moving, let M be the index set of all its resources, and let P be the empty set. If it is moving, let M be the index set of its independently moving resources, and let P be the index set of its passenger resources (which may be empty). (These terms are defined in Section 4.) Let pc be the unit's posture class. Then, if s = 1,

$$D(ibu,k) = \sum_{i \in M} tframe * ssvncr(k,i,pc) * [resources](ibu,i)$$

$$+ \sum_{i \in P} tframe * ssvncr(k,i,1) * [resources](ibu,i);$$

if s = 2, the expression for D(ibu,k) is the same except that *ssvncb* replaces *ssvncr*. (A sum over the empty set is defined to be 0.) Thus, every resource demands supplies according to its unit's posture class, except that passenger resources consume supplies as though in posture class 1.

Case 2: unit ibu is engaged. Let pp be its posture, and let

$$p = \begin{cases} pp - 19; & pp \geq 40 \\ pp - 9; & pp < 40. \end{cases}$$

Let

$$\text{index} = \mathit{mapps}(\text{s,p}).$$

For every $1 \leq \text{irs} \leq \text{nrs(s)}$, let

$$\text{lambda(irs)} = \text{frinv(irs,ibu)},$$

the fraction of the unit's resources of type irs that are actively involved in combat. Then

$$\begin{aligned} D(\text{ibu},k) = &\sum_{\text{irs}=1}^{\text{nrs(s)}} \mathit{tframe} * \mathit{ssvact}(\text{k,irs,index}) \\ &\quad * (\text{lambda(irs)} * [\text{resources}](\text{ibu,irs})) \\ + &\sum_{\text{irs}=1}^{\text{nrs(s)}} \mathit{tframe} * \mathit{ssvres}(\text{k,irs,index}) \\ &\quad * (1 - \text{lambda(irs)}) * [\text{resources}](\text{ibu,irs}). \end{aligned}$$

Because the rate of supplies consumption might depend strongly on the attack or defense posture, the game design variables *ssvact* and *ssvres* can distinguish different attack or defense postures. The variable *mapps*, which induces the third subscript of *ssvact* and *ssvres*, can be used to consolidate attack postures (40-49) or defense postures (10-29), thereby reducing the storage requirements of *ssvact* and *ssvres*.

The preceding defines any battle unit's demands for supplies. Again choose a side s battle unit, unit ibu. Suppose it does not belong to a task force. Let $1 \leq k \leq \mathit{nss}(s)$. The unit's present stock of type k supplies, stk, is given by

$$\text{stk} = [\text{resources}](\text{ibu,nequip(s)+k}).$$

The quantity of type k supplies it consumes is computed as

$$C = \min\,\{D(\text{ibu},k),\ \text{stk}\}$$

(it cannot consume more than it has), and therefore its stock of type k supplies at the end of the frame is redefined as follows:

$$[\text{resources}](\text{ibu,nequip(s)+k}) \longleftarrow \text{stk} - C.$$

Alternatively, suppose unit ibu is an element of a task force (possibly the only element). Let TF be the set of every unit in the task force, identified by unit number. Choose $1 \leq k \leq nss(s)$. The goal is to determine how much of the type k supplies held by unit ibu are consumed in the frame. The task force's total demand for type k supplies is

$$dd = \sum_{i \varepsilon TF} D(i,k).$$

Its total stock of type k supplies is

$$stk = \sum_{i \varepsilon TF} [resources](i,nequip(s)+k).$$

The amount of type k supplies consumed by the task force is computed as

$$C = \min \{dd, stk\}.$$

Each element of the task force is assessed the same fraction of its stock of type k supplies:

$$[resources](i,nequip(s)+k) \leftarrow \frac{stk - C}{stk} * [resources](i,nequip(s)+k)$$

for every $i \varepsilon$ TF and, in particular, for i = ibu. Thus, the elements of a task force share their supplies.

After assessing supplies consumption by a task force in a movement posture, IDAHEX ascertains whether the task force has exhausted its supplies of any type (assuming $nss(s) > 0$). If so, the task force may lack supplies it needs in order to move and should not be allowed to change location. IDAHEX finds what its movement delay would be if it were just starting its movement, in its present posture. If that delay equals or exceeds 10**9, the task force's mission is changed to a single order specifying 10 as the desired posture and its present location as the desired objective--which causes the task force to abort its movement and attempt to revert to a hold posture in its present location.

8.2 DELIVERING SUPPLIES AND REPLACEMENTS

IDAHEX can be used to represent the process by which personnel and equipment replacements, as well as supplies, are delivered to consuming units. The game designer makes this possible by

providing each side with one or more units representing depots and a fairly large number of units representing transport units. A player assigns missions to his transport units causing them to move back and forth between the depots and consuming units; he uses transfer and delivery commands to transfer resources from depots to transport units and from transport units to consuming units. Since the transport units are actually moving back and forth, the side's ability to sustain its units in combat will depend upon the quantities and types of transport that the transport units have and the lengths and trafficability of the lines of supply. And like any battle unit, transport units may be attacked from the air or ground.

Alternatively, the game designer may choose to provide each side with one or more depots but without any transport units. A player can use the send command to transfer resources from depots to distant units. Of course, his use of this command must be constrained by rules and judgments imposed outside IDAHEX; at a minimum, the rules should prevent overland resupply of an encircled unit. This approach is advantageous because it retains logistics constraints on a side without burdening the player with the task of controlling a large number of transport units.

8.3 MAINTENANCE

Maintenance is modeled only to the extent of representing repair of damaged equipment. Maintenance is played if and only if the logical variable *ifrep* has the value .true.. Maintenance assessment, if any, is performed by the entry point repair, which is called at the end of each frame, immediately after ground combat.

Equipment to be repaired is segregated into two "repair classes". Within each repair class, equipment is segregated according to the battle unit that owns it--the unit that held it when it developed a requirement for repairs, and that will get it back when the repairs are completed. Equipment awaiting repair is recorded in the matrix reppool. Equipment in repair class 1 is recorded in columns 1 through ilrep2-1; equipment in repair class 2 is recorded in the remaining columns.[1] If, for some i ≥ ilrep2, repbu(i) > 0, then reppool(ieq,i) is the quantity of type ieq equipment from unit repbu(i) in repair class 2. If repbu(i) ≤ 0, column i of reppool contains no information.

When all of a battle unit's equipment in repair class 1 has been repaired, repbu(i) is immediately set to 0 for that i ≥ ilrep2 such that repbu(i) equals the unit's number. When all of a unit's equipment in repair class 2 has been repaired,

---

[1]The cutoff point, ilrep2, is set by the entry point cgcm.

repbu(i) is immediately set to 0 for that i $\geq$ ilrep2 such that repbu(i) equals the unit's number. Also, when a unit is destroyed, any entries in repbu corresponding to it are immediately set to 0 (by the entry point xeq); hence, damaged equipment is automatically lost when the unit that owns it is destroyed.

Suppose unit ibu owns equipment in repair class 1. The equipment repaired in the frame must be determined. Let E(j) be the quantity of the unit's type j equipment in repair class 1. Let s = 1 if the unit is Red and s = 2 if it is Blue. Let n = nequip(s). The effort needed to repair all the equipment is computed as

$$D = \sum_{j=1}^{n} \mathit{repdd}(i,1,s) * E(j).$$

The game design datum *repdd*(i,1,s) measures the effort needed to repair one item (more precisely, a unit-quantity) of side s type i equipment in repair class 1. The repair capability is computed as

$$C = \mathit{tframe} * \mathit{repcu}(\mathit{butype}(\mathrm{ibu})).$$

The repair capability is allocated to each type of equipment in proportion to the repair effort it demands, with the result that the quantity of type j equipment repaired is

$$R(j) = \min\,\{C/D,\ 1\} * E(j).$$

The quantity of the battle unit's type j equipment in repair class 1 is reduced by R(j) for each $1 \leq j \leq n$. (The appropriate column of reppool is redefined.) And the unit's stock of equipment is increased:

$$[\mathrm{resources}](\mathrm{ibu},j) \longleftarrow [\mathrm{resources}](\mathrm{ibu},j) + R(j)$$

for every $1 \leq j \leq n$.

One rationale for the preceding procedure--but not the only one--is that repair class 1 consists of equipment that is lightly or moderately damaged and can be repaired by the battle unit that owns it. The datum *repcu*(k) gives the repair capability (per unit-time) of a type k battle unit.

Correspondingly, one may rationalize that repair class 2 consists of equipment that is severely damaged and can only be repaired in rear-area maintenance facilities. The game design datum *repcs*(s), for s = 1 or s = 2, is a measure of the aggregate capability of side s to repair equipment in repair class 2 (per unit-time); this capability is assumed to be independent of the

effort spent on repairing equipment in repair class 1. Despite the rationale that equipment in repair class 2 is repaired in rear-area depots, repaired equipment is returned to the battle unit from which it came. The alternative of pooling repaired equipment and letting the player decide which battle unit should receive it would give a side too much freedom to relocate equipment, implying that the side might even benefit from having equipment damaged.

When a unit without any equipment in repair class 2 loses equipment to that class, the time at which the equipment is lost is recorded in rept: the damaged equipment is recorded in reppool(*,i) for some i ≥ i1rep2, the battle unit's number is recorded in repbu(i), and the time is recorded in repbu(i-i1rep2+1). As long as the unit continues to have equipment in repair class 2, rept(i-i1rep2+1) remains the same. Thus rept(i-i1rep2+1) is the earliest time at which some of the unit's equipment currently in repair class 2 might have suffered the damage. This time--the unit's "repair request time"--is used to determine the unit's priority with respect to other units in getting its equipment from the depot; the unit takes precedence over units with later repair request times.

Let s = 1 or s = 2. The task is to assess the repair of side s equipment in repair class 2. The side's repair capability in the frame is

$$C = \mathit{tframe} * \mathit{repcs}(s).$$

The procedure is as follows:

<u>Step 0</u>. If C ≤ 0, go to Step 4.

<u>Step 1</u>. Of the side s battle units with equipment in repair class 2, find the one whose repair request time is earliest; let ibu be the unit's number. Let E(j) be the quantity of the unit's equipment in repair class 2 (1 ≤ j ≤ nequip(s)). The effort needed to repair this equipment is

$$D = \sum_{j=1}^{n} \mathit{repdd}(j,2,s) * E(j),$$

where n = nequip(s). Let

$$f = \min \{C/D, 1\}.$$

Let

$$R(j) = f * E(j)$$

for every j; R(j) is the quantity of the unit's equipment in repair class 2 on which repairs are completed.

Step 3. Reduce C to reflect the repair capability expended:

$$C \leftarrow C - D.$$

For each $1 \leq j \leq n$, reduce the unit's type j equipment in repair class 2 by f * E(j), and increase the unit's stock:

$$[\text{resources}](\text{ibu},j) \leftarrow [\text{resources}](\text{ibu},j) + f * E(j).$$

Go to Step 0.

Step 4. End.

The preceding procedure helps minimize the number of battle units with entries in reppool. If the number grew too large, some units' damaged equipment could not be recorded (due to lack of space in the array reppool) and therefore would be permanently lost; in that event IDAHEX would place a warning in the game designer's output file.

9. DAMAGE, REPAIR, AND IMPROVEMENT OF LINES OF COMMUNICATION

Ground resources and air-to-ground weapons may render sections of rail links unusable by, for example, tearing up or deforming the tracks. Such activity is termed *railroad destruction*. Battle units and air strikes may render sections of road links unusable by, for example, destroying the roadbed or laying mines. Such activity is termed *road destruction*. Battle units may repair damage to rail or road links resulting from rail or road destruction activity. Battle units and air strikes may make barriers harder to cross: they may, for example, destroy bridges over rivers or defiles, precipitate landslides in mountain passes, and lay mine belts. Such activity is termed *barrier intensification*. Battle units may counter barrier intensification efforts: for example, they may repair bridges, clear landslides, and clear mine belts. Such activity is termed *barrier de-intensification*. Battle units may make barriers easier to cross: for example, they may build bridges over rivers or defiles, smash walls, and tunnel through ridges. Such activity is termed *barrier mitigation*. Road destruction, railroad destruction, barrier intensification, road repair, railroad repair, barrier de-intensification, and barrier mitigation collectively are termed LOC (line of communication) modification activities.

Suppose the road or rail link between two adjacent cells, i and j, is damaged, or the barrier between them is intensified or mitigated, as the result of an LOC modification activity. Information about the current status of the LOC between the cells is recorded in the LOC Effects List, whose columns are the vectors celoc1, celoc2, cetype, and celvl. Specifically, for some $k \geq 1$, celoc1(k) equals the lesser of i and j, and celoc2(k) equals the greater of i and j; then cetype(k) indicates what type of effect exists, and celvl(k) indicates how great the effect is. The datum cetype(k) is coded as follows:

cetype(k)	type of LOC effect
1	barrier intensification
2	railroad damage
3	road damage
4	barrier mitigation

If cetype(k) = 2 or cetype(k) = 3, celvl(k) is the total length of the road or rail link between cells i and j that is unusable; the link's total length is assumed to equal *depth*, the distance between the cells' centers. If cetype(k) = 1 or cetype(k) = 4, celvl(k) is a less direct measure of the effect on the barrier, as explained below.

9.1 LOC EFFECTS AND TRAFFICABILITY

9.1.1 Roads and Railroads

Recall from Section 3 that resources trying to move by rail suffer an infinite delay if the rail link between their location and their objective is damaged, and resources trying to move entirely by road or by some combination of road and cross-country movement must move off-road to the extent that the road link is damaged. Thus, calculating movement rates may require knowledge of what fraction of the road or rail link between two adjacent cells is unusable due to damage.

The unusable, or damaged, fraction of the rail link between two adjacent cells, i and j, is found as follows:

Step 0. If there is no rail link between cells i and j--i.e., if [rail](i,j) = 0--then define the unusable fraction to be 0 and go to Step 2.

Step 1. Let m = min(i,j) and n = max(i,j). Search for k such that celoc1(k) = i, celoc2(k) = j, and cetype(k) = 2. If no such k exists, define the unusable fraction to be 0 and go to Step 2. Define the unusable fraction to be

celvl(k) / *depth*.

Step 2. End.

The unusable, or damaged, fraction of the road link between adjacent cells i and j is found as follows:

Step 0. If there is no road link between cells i and j--i.e., if [road](i,j) = 0--then define the unusable fraction to be 0 and go to Step 2.

Step 1. Let m = min(i,j) and n = max(i,j). Search for k such that celoc1(k) = i, celoc2(k) = j, and cetype(k) = 3. If no such k exists, define the unusable fraction to be 0 and go to Step 2. Define the unusable fraction to be

celvl(k) / *depth*.

Step 2. End.

9.1.2 Barriers

Barrier intensification may alter the effective type of barrier between two cells. Let cell i and cell j be adjacent. The type of movement barrier between them, movebar(i,j), and the type of attack barrier, atkbar(i,j), are found as follows:

Step 0. Let bt0 = [basic_barrier](i,j). If there is no barrier between cell i and cell j, i.e., if bt0 = 0, then set

$$\text{movebar}(i,j) = 0,$$
$$\text{atkbar}(i,j) = 0,$$

and go to Step 3.

Step 1. Let m = min(i,j) and n = max(i,j). Search for k such that celoc1(k) = m, celoc2(k) = n, and cetype(k) = 1. If no such k exists, set

$$\text{movebar}(i,j) = \mathit{mapmb}(\text{bt0}),$$
$$\text{atkbar}(i,j) = \mathit{mapab}(\text{bt0}),$$

and go to Step 3. Let

$$\text{fx} = \text{celvl}(k),$$
$$\text{bt} = \text{bt0}.$$

Step 2. If

$$\text{fx} > \mathit{bistep}(\text{bt}),$$

or if $\mathit{bifx}(\text{bt}) = \text{bt}$, then set

$$\text{movebar}(i,j) = \mathit{mapmb}(\text{bt}),$$
$$\text{atkbar}(i,j) = \mathit{mapab}(\text{bt}),$$

and go to Step 3. Reduce fx by the step size:

$$\text{fx} \leftarrow \text{fx} - \mathit{bistep}(\text{bt}).$$

Redefine the barrier type:

$$\text{bt} \leftarrow \mathit{bifx}(\text{bt}).$$

Repeat Step 2.

Step 3. End.

The game design datum *bistep*(k) measures the quantity of effort that must be expended in barrier intensification in order to transform a type k barrier into another type; the design datum *bifx*(k) is the type of barrier it becomes. Notice that only a barrier whose basic type (given by [basic_barrier]) is positive can be intensified; if barrier intensification should be possible everywhere (which would be the case if the game designer wanted to allow mine belts to be laid between any two adjacent cells), [basic_barrier](i,j) should be positive for every pair of adjacent, active cells, i and j.

9.2 BATTLE UNITS' LOC MODIFICATION ACTIVITIES AND LOC EFFECTS

The preceding subsection shows how information from the LOC Effects List is used to determine road usability, railroad usability, and barrier types. This subsection shows how battle units' LOC modification activities create LOC effects, with corresponding changes in the LOC Effects List and in basic barrier types. The following subsection shows how air forces' LOC modification activities create LOC effects.

Battle units' LOC modification activities are recorded in the Special Activities List, which contains information about units' activities that may not be discernible from their postures alone. A line of the List describes an ongoing activity of a single battle unit. At the end of each frame, after supplies consumption is assessed, the List is searched, and the consequences of each LOC modification activity are assessed; these may include LOC effects represented by entries in the LOC Effects List, and attendant supplies consumption. The List's columns are the vectors saloc1, saloc2, satype, saunit, and salvl; saloc1(k) = 0 signifies that line k of the List is unused. If saloc1(k) > 0, satype(k) indicates the type of special activity specified by line k of the List:

satype(k)	special activity
1	barrier intensification
2	railroad destruction
3	road destruction
4	barrier de-intensification
5	railroad repair
6	road repair
7	barrier mitigation
10	close supporting fire

The game designer may decide not to play some, or all, special activities in order to reduce design data requirements. IDAHEX permits side s battle units (s=1 or s=2) to engage in special activity k, and demands design data concerning it, if and only if *ifsa*(k,s) = .true.. (*ifsa* is a logical game design vector.)

Suppose, for some $k \geq 1$, that saloc1(k) > 0 and $1 \leq$ satype(k) ≤ 7. Then line k describes an ongoing LOC modification activity of battle unit saunit(k). Let

$$\begin{aligned} \text{isat} &= \text{satype(k)} \\ \text{loc1} &= \text{saloc1(k)} \\ \text{loc2} &= \text{saloc2(k)} \\ \text{m} &= \text{min (loc1, loc2)} \\ \text{n} &= \text{max (loc1, loc2)} \\ \text{ibu} &= \text{saunit(k)} \\ \text{q} &= \text{salvl(k)} \end{aligned}$$

The List indicates that unit ibu is to perform LOC modification on the LOC between cells m and n, which should be adjacent. The datum q, which should be in the interval (0,1], may be interpreted as the fraction of the unit's resources to participate in the special activity; a possible reason for making the fraction less than 1 is to conserve supplies.

For $1 \leq \ell \leq$ *nroad*, the design datum *desreq*(ℓ) is a relative measure of the effort needed to make a unit-length of type ℓ road link unusable, and *repreq*(ℓ) is a relative measure of the effort needed to repair it. For $1 \leq \ell \leq$ *nrail*, *desreq*(*nroad* +ℓ) is a relative measure of the effort needed to make a unit-length of type ℓ rail link unusable, and *repreq*(ℓ) is a relative measure of the effort needed to repair it.

A battle unit's capability to do road or rail destruction or road or rail repair depends on its resources and *cecap*: for s = 1 or s = 2, the game design datum *cecap*(irsoff(s)+irs, isat) is a measure of the capability of side s type irs resources to perform special activity isat. But the datum may be irrelevant if the LOC modification activity is barrier intensification, de-intensification, or mitigation: if--for isat = 1, isat = 4, or isat = 7--*locmop*(isat) = 0, then a battle unit's output in special activity isat depends only on how long it performs the activity, and not on its resources or *cecap*. This option is useful since tasks such as bridge destruction and bridge construction may be constrained more by time (due to their sequential nature) than by the availability of resources.

The previously identified special activity's consequences in a single frame are assessed as follows:

Step 0. Unless battle unit ibu is located in cell i or cell j and is active, go to Step 6. (No LOC modification occurs.) Unless the unit's side owns its location, go to Step 6. (LOC modification by ground forces is prohibited unless the enemy might be able to prevent it.) Let s = 1 if unit ibu is Red, and s = 2 if it is Blue. Supplies consumption resulting from the special activity is assessed in Step 5, but unless unit

ibu has the supplies the activity demands, the activity may not occur. If, for some $1 \leq$ kss $\leq$ *nss*(s),

$$[\text{resources}](\text{ibu},\text{nequip}(s)+\text{kss}) = 0$$

and $\sum_{\text{irs}=1}^{\text{nrs}(s)}$ [resources](ibu,irs) * *ssreqe*(kss,irs,isat,s) > 0,

then go to Step 6. Let

$$\text{i0} = \text{irsoff}(s).$$

Go to

Step 1 if isat = 1,
Step 2 if isat = 2 or isat = 3,
Step 3 if isat = 4 or isat = 7,
Step 4 if isat = 5 or isat = 6.

Step 1 (barrier intensification). Search the LOC Effects List for a relevant entry; to be precise, search for a value of the variable ice such that

celocl(ice) = m,
celoc2(ice) = n,
cetype(ice) = isat.

If no such value of ice exists, let ice be the number of any blank line in the LOC Effects List, and initialize:

celocl(ice) ← m,
celoc2(ice) ← n,
cetype(ice) ← isat,
celvl(ice) ← 0.

If *locmop*(isat) $\neq$ 0, let

$$\text{cap} = \sum_{\text{irs}=1}^{\text{nrs}(s)} \mathit{cecap}(\text{i0}+\text{irs},\text{isat}) * [\text{resources}](\text{ibu},\text{irs}).$$

If *locmop*(isat) = 0, let

$$\text{cap} = 1.$$

Redefine celvl(ice):

$$\text{celvl(ice)} \leftarrow \text{celvl(ice)} + \mathit{tframe} * q * \text{cap}.$$

Let fx = celvl(ice). Let bt = [basic_barrier](m,n).

ILOOP: If *bifx*(bt) = bt or fx < *bistep*(bt), go to Step 5. Redefine fx and bt:

$$fx \leftarrow fx - \mathit{bistep}(bt),$$
$$bt \leftarrow \mathit{bifx}(bt).$$

If *biperm*(bt) = .false., go back to ILOOP. Redefine the basic barrier type:

$$[\text{basic\_barrier}](m,n) \leftarrow bt.$$

Redefine celvl(ice) to reflect the effort expended in transforming the basic barrier type:

$$\text{celvl}(ice) \leftarrow fx.$$

Go to ILOOP.

Notice this step's resemblance to Step 2 in Section 8.1.2. Both steps yield the same value of bt, but this step may change the basic barrier type in the process. The key variable in deciding whether to change the basic barrier type is *biperm*(bt), a logical game design variable; *biperm*(bt) = .true. signifies that barrier intensification yielding barrier type bt is too serious to be negated by barrier de-intensification. For example, a highway bridge over a river might be damaged so severely that building a new bridge would be easier than repairing the old one. Go to Step 5.

<u>Step 2</u> (rail or road destruction). If isat = 2 and [rail](n,m) = 0, go to Step 6. If isat = 3 and [road](m,n) = 0, go to Step 6. Let

$$\text{cap} = \sum_{irs=1}^{nrs(s)} \mathit{cecap}(i0+irs,isat) * [\text{resources}](ibu,irs).$$

Let

$$\text{iat} = \begin{cases} [\text{road}](m,n) \text{ if isat} = 3, \\ \mathit{nroad} + [\text{rail}](m,n) \text{ if isat} = 2. \end{cases}$$

Let

$$\text{delta} = \mathit{tframe} * q * (\text{cap} / \mathit{desreq}(\text{iat})).$$

The number delta is an estimate of the length of road or rail link destroyed in the frame by unit ibu, but unless the damage is significant, it is ignored. The test of significance, which must be passed for an LOC effect to be recorded, is

whether delta exceeds a threshold fixed by the game designer: if

$$\text{delta} < \textit{deseps}(\text{iat}),$$

go to Step 6. Search for a value of ice such that

$$\begin{aligned} \text{celoc1(ice)} &= \text{m}, \\ \text{celoc2(ice)} &= \text{n}, \\ \text{cetype(ice)} &= \text{isat}. \end{aligned}$$

If no such value exists, let ice be the number of any blank line in the LOC Effects List, and initialize:

$$\begin{aligned} \text{celoc1(ice)} &\leftarrow \text{m}, \\ \text{celoc2(ice)} &\leftarrow \text{n}, \\ \text{cetype(ice)} &\leftarrow \text{isat}, \\ \text{celvl(ice)} &\leftarrow 0. \end{aligned}$$

Redefine celvl(ice) to reflect the damage:

$$\text{celvl(ice)} \leftarrow \min\{\text{celvl(ice)} + \text{delta}, \textit{depth}\}.$$

If celvl(ice) = *depth*, then set saloc1(k) = 0, terminating the special activity. Go to Step 5.

Step 3 (barrier de-intensification or mitigation). Let

$$\text{bt} = [\text{basic\_barrier}](\text{m,n}).$$

If bt ≤ 0, go to Step 6. Search for a value of ice such that

$$\begin{aligned} \text{celoc1(ice)} &= \text{m}, \\ \text{celoc2(ice)} &= \text{n}, \\ \text{cetype(ice)} &= 1. \end{aligned}$$

If no such value exists, let F = 0 and go to Step 3(b).

Step 3(a) (barrier de-intensification). Let temp = celvl(ice). If *locmop*(4) ≠ 0, let

$$\text{cap} = \sum_{\text{irs}=1}^{\text{nrs(s)}} \textit{cecap}(\text{i0+irs},4) * [\text{resources}](\text{ibu,irs}).$$

If *locmop*(4) = 0, let

$$\text{cap} = 1.$$

Redefine celvl(ice) to reflect the de-intensification:

$$\text{celvl(ice)} \leftarrow \text{celvl(ice)} - \textit{tframe} * \text{q} * \text{cap}.$$

Let

$$F = \min \{\text{temp}/(\mathit{tframe} * q * \text{cap}), 1\}$$

if temp > 0 and cap > 0; let F = 0 otherwise. If celvl(ice) ≤ 0, set celocl(ice) = 0 (in effect, obliterating line ice of the LOC Effects List). If isat ≠ 7, go to Step 5.

Step 3(b) (barrier mitigation). If isat ≠ 7 go to Step 6. Search for a value of ice such that

celocl(ice) = m,
celoc2(ice) = n,
cetype(ice) = 4.

If none exists, let ice be the number of any blank line in the LOC Effects List, and initialize:

celocl(ice) ← m,
celoc2(ice) ← n,
cetype(ice) ← 4,
celvl(ice) ← 0.

If *locmop*(isat) ≠ 0, let

$$\text{cap} = \sum_{\text{irs}=1}^{\text{nrs(s)}} \mathit{cecap}(\text{10+irs,isat}) * [\text{resources}](\text{ibu,irs}).$$

If *locmop*(isat) = 0, let

$$\text{cap} = 1.$$

Let

$$\text{fx} = (1-F) * (\mathit{tframe} * q * \text{cap}).$$

Redefine celvl(ice):

celvl(ice) ← celvl(ice) + fx.

LOOP: Let bt = [basic_barrier](m,n). If

celvl(ice) < *bmstep*(bt),

go to Step 5. If *bmfx*(bt) = bt, then set celocl(ice) = 0, set salocl(k) = 0 (terminating the special activity) and go to Step 5. Redefine celvl(ice) and transform the basic barrier type:

celvl(ice) ← celvl(ice) - *bmstep*(bt),
[basic_barrier](m,n) ← *bmfx*(bt).

($bmfx$(bt) is the basic barrier type that results when a barrier of basic type bt is subjected to a level of barrier mitigation effort equal to $bmstep$(bt).) Go back to LOOP.

Step 4 (rail or road repair). If isat = 5 and [rail](m,n) = 0, go to Step 6. If isat = 6 and [road](m,n) = 0, go to Step 6. Search for a value of ice such that

$$\begin{aligned} \text{celocl(ice)} &= \text{m}, \\ \text{celoc2(ice)} &= \text{n}, \\ \text{cetype(ice)} &= \text{isat} - 3. \end{aligned}$$

If none exists, go to Step 6 Let

$$\text{cap} = \sum_{\text{irs}=1}^{\text{nrs(s)}} \mathit{cecap}(\text{10+irs,isat}) * [\text{resources}](\text{ibu,irs}).$$

Let

$$\text{isat} = \begin{cases} [\text{road}](\text{m,n}) \text{ if isat} = 6, \\ \mathit{nroad} + [\text{rail}](\text{m,n}) \text{ if isat} = 5. \end{cases}$$

Let

$$\text{delta} = \mathit{tframe} * \text{q} * \text{cap} \;/\; \mathit{repreq}(\text{iat}).$$

This is the length of the road or rail link repaired in the frame by unit ibu. Redefine celvl(ice) accordingly:

$$\text{celvl(ice)} \leftarrow \text{celvl(ice)} - \text{delta}.$$

If celvl(ice) $\leq$ 0, then set celocl(ice) = 0, and also set salocl(k) = 0, terminating the special activity. Go to Step 5.

Step 5. The rate of consumption of type kss supplies by side s type irs resources performing special activity ℓ is, by definition, $ssreqe$(kss,irs,ℓ,s). Supplies consumption resulting from the special activity is assessed according to the following procedure. Note that this consumption is in addition to the routine supplies consumption, assessed as Section 7 describes. For each 1 $\leq$ kss $\leq$ nss(s), let

$$\text{CR} = \sum_{\text{irs}=1}^{\text{nrs(s)}} \mathit{ssreqe}(\text{kss,irs,isat,s}) * [\text{resources}](\text{ibu,irs}),$$

let delta = $tframe$ * q * CR,

and finally, reduce the unit's supplies:

[resources](ibu,nequip(s)+kss)

← max ([resources](ibu,nequip(s)+kss) - delta, 0).

<u>Step 6</u>. End.

9.3 LOC EFFECTS OF AIR STRIKES

This subsection shows how air-to-ground weapons employed for the purpose of LOC modification create LOC effects, with corresponding changes in the LOC Effects List. Whereas a battle unit's LOC modification activity is a process, whose parameters must be recorded in the Special Activities List and whose LOC effects must be assessed periodically, air-to-ground weapons' LOC effects are assessed once-and-for-all at the time of the air strike.

The concept analogous to a battle unit's special activity is an air strike's role. The strike roles that imply LOC effects are: barrier intensification, railroad destruction, and road destruction. No strike roles correspond to the other LOC modification activities: air strikes cannot de-intensify barriers, repair railroads, repair roads, or mitigate barriers. Strike roles are coded as follows, to correspond to the special activity codes:

<u>role number</u>	<u>role name</u>
1	barrier intensification
2	railroad destruction
3	road destruction

The game designer may decide to exclude some or all strike roles in order to reduce design data requirements. IDAHEX permits a side s air strike (s=1 or s=2) to have role k, and demands design data concerning it, if and only if *ifrole*(k,s) = .true.. (*ifrole* is a logical game design vector.)

Let s = 1 or s = 2. Let i0 = 0 if s = 1, and i0 = *nagwep*(1) if s = 2. The game design datum *cecapa*(i0+iw,k) is a measure of the capability of a side s type iw air-to-ground weapon in strike role k, where $1 \leq iw \leq$ *nagwep*(s) and $1 \leq k \leq 3$. For $1 \leq i \leq$ *nroad*, *desrqa*(i) is a relative measure of the difficulty of making a unit-length of type i road link unusable by air attack. For $1 \leq i \leq$ *nrail*, *desrqa*(*nroad*+i) is a relative measure of the difficulty of making a unit-length of type i rail link unusable by air attack.

Suppose there is an air attack by side s for the purpose of damaging a rail link, damaging a road link, or intensifying a barrier between two adjacent, active cells, loc1 and loc2. Let

$$m = \min(\text{loc1}, \text{loc2}),$$
$$n = \max(\text{loc1}, \text{loc2}).$$

Let agwep(i) be the quantity of type i air-to-ground weapons used; it is computed from the strike composition, as Section 7 explains. Let i0 = 0 if s = 1, and i0 = *nagwep*(1) if s = 2. Let nw = *nagwep*(s). Let irole be the numerical code of the strike role.

<u>Step 0</u>. Go to Step 1 if the strike role is barrier intensification; go to Step 2 if it is railroad destruction or road destruction.

<u>Step 1</u> (barrier intensification). Search the LOC Effects List for a relevant entry; to be precise, search for a value of the variable ice such that

$$\text{celoc1(ice)} = m,$$
$$\text{celoc2(ice)} = n,$$
$$\text{cetype(ice)} = \text{irole}.$$

If none exists, let ice be the number of any blank line in the LOC Effects List, and initialize:

$$\text{celoc1(ice)} \leftarrow m,$$
$$\text{celoc2(ice)} \leftarrow n,$$
$$\text{cetype(ice)} \leftarrow \text{irole},$$
$$\text{celvl(ice)} \leftarrow 0.$$

Let

$$fx = \sum_{iw=1}^{nw} \mathit{cecapa}(\text{i0+iw,irole}) * \text{agwep(iw)}.$$

Redefine celvl(ice) to include the strike's effects:

$$\text{celvl(ice)} \leftarrow \text{celvl(ice)} + fx.$$

Go to Step 3.

<u>Step 2</u> (rail or road destruction). If irole = 2 and [rail](m,n) = 0, go to Step 3. If irole = 3 and [road](m,n) = 0, go to Step 3. Let

$$fx = \sum_{iw=1}^{nw} \mathit{cecapa}(i0+iw,irole) * agwep(iw).$$

Let

$$iat = \begin{cases} [road](m,n) \text{ if } irole = 3, \\ \mathit{nroad} + [rail](m,n) \text{ if } irole = 2. \end{cases}$$

Let

$$delta = fx \;/\; \mathit{desrqa}(iat).$$

This is an estimate of the length of road or rail link destroyed by the air strike, but unless the damage is significant, it is ignored. The test of the significance, which must be passed for an LOC effect to be recorded, is whether delta exceeds a threshold fixed by the game designer: if

$$delta < \mathit{desepa}(iat),$$

go to Step 3. Search for a value of ice such that

$$\begin{aligned} celoc1(ice) &= m, \\ celoc2(ice) &= n, \\ cetype(ice) &= irole. \end{aligned}$$

If none exists, let ice be the number of any blank line in the LOC Effects List, and initialize:

$$\begin{aligned} celoc1(ice) &\leftarrow m, \\ celoc2(ice) &\leftarrow n, \\ cetype(ice) &\leftarrow irole, \\ celvl(ice) &\leftarrow 0. \end{aligned}$$

Redefine celvl(ice) to reflect the damage:

$$celvl(ice) \leftarrow \min \{celvl(ice) + delta, \mathit{depth}\}.$$

Step 3. End.

10. COMMUNICATING WITH THE IDAHEX COMPUTER PROGRAM

IDAHEX uses the following files:

file10 - Red player input
file11 - Red player output
file20 - Blue player input
file21 - Blue player output
file50 - game design (input) data
file51 - game designer's output file
file60 - game design (input) data

The program references a file by using its number--10, 11, 20, 21, 50, 51, or 60--as the data set reference number in a FORTRAN formatted read or write statement or by using its name (file10, file11, etc.) as the file name in a PL/I get or put statement.

File 50 contains all the game design data except the values of *mapter, maprd, maprr, mapmb,* and *mapcb*. File 60 contains the data that define *mapter, maprd, maprr, mapmb,* and *mapcb* in each cycle. The format and sequence of the data in file 50 and file 60 are explained in Section 11. File 51 contains IDAHEX's interpretation of the data in file 50, and warning or error messages if IDAHEX questions the correctness of the data. An error message indicates that IDAHEX was unable to interpret the input data. It may continue processing the game design data, but it will terminate execution before the players can enter commands. A warning draws the game designer's attention to a possible error in the design data; execution continues. If execution is allowed to proceed and a game is played, file 51 also contains a description of the game's events.

The game design datum *nprint* indicates the number of distinct data sets that are being used. If *nprint* = 1, IDAHEX expects files 50, 10, and 20 to be associated with the same data set (usually card reader input) and all the output files to be associated with the same data set (usually high speed printer output). If *nprint* = 2, IDAHEX expects file 50 to be associated with a different data set than files 10 and 20, which it expects to be associated with the same data set, and it expects file 51 to be associated with a different data set than files 11 and 21, which it expects to be associated with the same data set. If *nprint* = 3, IDAHEX expects every file to be associated with a

different data set. No matter what the value of *nprint*, file 60 must be associated with a distinct data set for which the rewind operation is permitted. Normally, *nprint* = 1 means that IDAHEX is being used in a batch processing mode; *nprint* = 2 means it is being used interactively with one terminal, which the players share; and *nprint* = 3 means it is being used with two terminals, one for the Red player and one for the Blue player.

By using the save command (see Volume 3, Section 4), a player can save the game situation in an unformatted, rewindable file that he designates by number. At least one file should be set aside for this purpose. It is wise to set aside more than one because, if not, every save will necessarily overwrite the previous one.

The file associations must be in effect when IDAHEX is invoked. The following MULTICS commands illustrate how the file associations are established when IDAHEX is to be played from exactly one terminal (*nprint* = 1).

```
io attach file10 syn_ user_input
io attach file11 syn_ user_output
io attach file20 syn_ user_input
io attach file21 syn_ user_output
io attach file50 vfile_ Sinai_dd
io attach file60 vfile_ Sinai_terrain_maps
io attach file90 vfile_ Sinai_dd_unformatted
io attach file91 vfile_ Sinai_game.1
io attach file92 vfile_ Sinai_game.2
io attach file93 vfile_ Sinai_game.3
set_cc file51 -on
set_cc file11 -on
set_cc file21 -on
line_length 115
```

The files 90, 91, 92, and 93 identified above are intended as places to save the game situation. The first character of every line output to files 11, 21, and 51 is a carriage control character; hence, the files' carriage control attribute is set to "on".

The IDAHEX main program is named cgcm. Invoking it invokes IDAHEX.

The game design variable *iprint* governs the output's level of detail. If *iprint* ≥ 1, file 51 will contain a complete description of every significant change in a battle unit's status. If *iprint* ≥ 5, the players will be informed of every significant change in a unit's status. If *iprint* ≥ 7, file 51 will contain a complete description of every change in a unit's

status. File 51 will always contain a detailed description of every engagement. If *iprint* ≥ 15, the players will receive the same description. If *iprint* < 15, they will not be informed of an engagement's average kill matrices (denoted A and D in Sections 6.1.1 and 6.1.2). If *iprint* < 9, they will not be informed of the values of the attackers' and defenders' weapons (Section 6.1.2). If *iprint* < 5, they will not be informed of the losses in the engagement. A value of 9 is generally best.

11. GAME DESIGN DATA INPUT

The game design data defining *mapter, maprd, maprr,* and *mapbar* are read from file 60. All the other game design data are read from file 50 at the beginning of a game, before play commences. File 50 consists of several sections. Each section contains the data read by a particular IDAHEX entry point. A section's first line (card image) is a marker; it contains the entry name without its "0" suffix. Lines coming between the end of a section and the next section's marker are ignored; they may be used to incorporate comments in file 50. Each section of file 50, as well as file 60 as a whole, consists of a sequence of groups. The groups are described below in the order in which they are read. The description of a group consists of: (1) a line listing the variables whose values the group fixes and (2) FORTRAN statements indicating how the group is read and therefore the correct order of the data within the group. The FORTRAN statements do not correspond exactly to IDAHEX source code, and although generally written according to MULTICS FORTRAN language conventions, are not necessarily valid source code for any compiler; their sole purpose is to explain how the contents of files 50 and 60 fix the values of the game design variables. Contrary to FORTRAN convention, the FORTRAN code in this section assumes that the statements in a do loop are not executed even once if the lower bound specified in the do statement exceeds the upper bound.

Game design variables' names are not italicized in this section. The only variables mentioned that are not game design variables are do loop indices and the following: nnsyl (fixed by cgcm), name, i, j, k, side, itemp, jtemp, ktemp, vtemp, temp, old, kap (defined in cmbt0), kdp (defined in cmbt0), nequip, nmat, nrs, nmarch (defined in wait0), and ntc (defined in wait0).

In accordance with the rest of the manual, some variables' names contain two components--for example, frinv.f, frinv.x, freff.f. Such a variable is referenced in only one subprogram; it takes the first component of its name from the subprogram's

name. In the actual IDAHEX source program, the variable's name is simply the second component of the two-component name used to identify it in this manual.

The following format statements are cited by many read statements in this section.

```
2 format (8i10)
3 format (8f10.0).
```

11.1 FILE 50

Each section must begin with a marker line containing the section name left-justified in columns 1 through 8. (In the sequel, the section name appears after the label "Section".) The sections may appear in any order if file 50 resides on a rewindable device; they must appear in the order indicated if not. The groups of data within a section must appear in the order indicated. The file's contents are indicated below:

Section: cgcm

1. iprint, nprint

```
read(50,2) iprint, nprint
```

Section: time

1. tinit, tend, tframe, tcycle, tpd, delta

```
read(50,3) tinit, tend, tframe, tcycle, tpd, delta
```

Section: net

1. ncells, nrankl

```
read(50,2) ncells, nrankl
```

2. ename0

```
1 read(50,4) i, (name(k), k = 1, nnsyl)
4 format (i5,5x,6a8)
  if (i.le.0) go to 6
  do 5 k = 1, nnsyl
     ename0(i,k) = name(k)
5    continue
```

```
      go to 1
    6 continue
```

3. nenv

```
      read(50,2) nenv
```

4. ename

```
      do 5 i = 1, nenv
      read(50,4) (ename(i,j), j = 1,nnsyl)
    4 format
    5 continue
```

5. [terrain]

```
      d0 6 i = 1, ncells
    6 [terrain](i) = 0
    1 read(50,2) i, itemp
      if (i.le.0) go to 5
      [terrain](i) = itemp
      go to 1
    5 continue
```

6. rname0

```
    1 read(50,4) i, (name(k), k = 1, nnsyl)
    4 format (i5,5x,6a8)
      if (i.le.0) go to 6
      do 5 k = 1,nnsyl
         rname0(i,k) = name(k)
    5    continue
      go to 1
    6 continue
```

7. rrnam0

```
    1 read(50,4) i, (name(k), k = 1, nnsyl)
    4 format (i5,5x,6a8)
      if (i.le.0) go to 6
      do 5 k = 1, nnsyl
         rrnam0(i,k) = name(k)
    5    continue
      go to 1
    6 continue
```

8. bname0

```
    1 read(50,4) i, (name(k), k = 1, nnsyl)
    4 format (i5,5x,6a8)
      if (i.le.0) go to 6
      do 5 k = 1, nnsyl
```

```
      bname0(i,k) = name(k)
    5   continue
      go to 1
    6 continue
```

9. nroad

```
      read(50,2) nroad
```

10. rname

```
      do 5 i = 1, nroad
    5    read(50,4) (rname(i,j), j = 1, nnsyl)
    4 format (6a8)
```

11. nrail

```
      read(50,2) nrail
```

12. rrname

```
      do 5 i = 1, nrail
    5 read(50,4) (rrname(i,j), j = 1, nnsyl)
    4 format (6a8)
```

13. nmb, nab

```
      read(50,2) nmb, nab
```

14. mbname

```
      do 5 i = 1, nmb
    5 read(50,4) (mbname)(i,j), j = 1, nnsyl)
    4 format (6a8)
```

15. abname

```
      do 5 i = 1, nab
    5 read(50,4) (abname(i,j), j = 1, nnsyl)
    4 format (6a8)
```

16. dirab

```
      do 5 i = 1, nab
    5 read(50,2) (dirab(i,j), j = 1, 6)
```

17. [basic_road], [basic_rail], [basic_barrier]

```
      do 6 i = 1, ncells
      do 6 j = 1, ncells
      [basic_road](i,j) = 0
      [basic_rail](i,j) = 0
```

```
      [basic_barrier](i,j) = 0
    1 read(50,2) i, j, itemp, jtemp, ktemp
      if (i.le.0) go to 5
      if (j.le.0) go to 4
      [basic_road](i,j) = itemp
      [basic_rail](i,j) = jtemp
      [basic_barrier](i,j) = ktemp
    4 continue
      go to 1
    5 continue
```

17. depth

```
      read(50,3) depth
```

Section: bu

1. iblul, nsyl, nutype

```
      read(50,2) iblul, nsyl, nutype
```

2. npost

```
      read(50,2) (npost(i), i = 1, 4)
```

3. itrfp

```
      read(50,2) itrfp
```

4. nggwep, ngawep, ntrpt, nss, npers

```
      do 5 i = 1, 2
    5 read(50,2) nggwep(i), ngawep(i), ntrpt(i), nss(i), npers(i)
```

5. rsname

```
      do 5 k = 1, 2
         do 5 i = 1, nrs(k)
            read(50,4) (rsname(i,j,k) j = 1, 2)
    4       format (a5,1x,a5)
    5       continue
```

6. flag

```
      read(50,2) (flag(i), i = 1, nutype)
```

7. nrst, iars

```
      do 5 i = 1, nutype
    5 read(50,2) nrst(i), (iars(j,i), j = 1, nrst(i))
```

8. toe

```
      do 5 i = 1, nutype
    1 read(50,3) (toe(i,j), j = 1, nrs(flag(i)))
```

9. aisize

```
      do 5 i = 1, nutype
    5 read(50,2) (aisize(i,j), j = 1, 4)
```

10. buname, butype, buloc, bupost, tentry, [resources]

```
    1 read(50,4) i, (vtemp(j), j = 1, nsyl)
    4 format (i5,5x,7a8)
      if (i.le.0) go to 10
      do 5 j = 1, nsyl
         buname(i,j) = vtemp(j)
    5    continue
      read(50,6) butype(i), buloc(i), bupost(i), tentry(i)
    6 format (3i10,f10.0)
      read(50,3) ([resources](i,j), j = 1, nrs(flag(butype(i)))
      go to 1
   10 continue
```

11. [owner]

```
      read(50,2) border, side
      do 5 i = 1, border
         [owner](i) = side
    5    continue
      side = 3 - side
      do 6 i = border + 1, ncells
         [owner](i) = side
    6    continue
    7 read(50,2) i, itemp
      if (i.le.0) go to 8
      [owner](i) = itemp
      go to 7
    8 continue
```

Section: wait

1. pmapup, pmapdn

```
      do 5 i = 10, 10
        pmapup(i) = 20
    5   pmapdn(i) = -10
      do 6 i = 20, 29
        pmapup(i) = 30
    6   pmapdn(i) = 40
      do 7 i = 30, 39
```

```
      pmapup(i) = 40
 7    pmapdn(i) = 40
    do 8 i = 40, 49
      pmapup(i) = 10
 8    pmapdn(i) = 40
10  rea  50,2) i, itemp, jtemp
    if (i.le.0) go to 11
    pmapup(i) = itemp
    pmapdn(i) = jtemp
    go to 10
11  continue
```

2. ptran

```
    do 5 i = 1, 4
       do 5 j = 1, npost(i)
 5        read(50,3) (ptran(i,j,k), k = 1, npost(i))
```

3. diseng

```
    read(50,3) (diseng(i), i = 1, nutype)
```

4. elude

```
    do 5 i = 1, 2
 5     read(50,3) (elude(j,i), j = 1, nrs(j))
```

5. mode

```
    do 5 j = 1, 2
 5     read(50,2) (mode(i,j), i = 1, nrs(j))
```

6. ldclas

```
    do 5 j = 1, 2
 5     read(50,2) (ldclas(i,j), i = 1, nrs(j))
```

7. fercl

```
    read(50,2) (fercl(i), i = 1, npost(3))
```

8. imclas

```
    do 5 j = 1, nutype
 5     read(50,2) (imclas(i,j), i = 1, npost(3))
```

9. vcc

```
    do 5 k = 1, 2
       do 5 i = 1, nrs(k)
 5        read(50,3) (vcc([irsoff](k) + i,j), j = 1, nenv)
```

10. fcc

```
      do 5 j = 1, 2
5        read(50,3) (fcc(i,j), i = 1, nmarch)
```

11. vroll

```
      do 5 k = 1, 2
         do 5 i = 1, nrs(k)
5           read(50,3) (vroll([irsoff](k) + i,j), j = 1, nroad + nrail)
```

12. frd

```
      do 5 k = 1, 2
         do 5 i = 1, nequip(k)
5           read(50,3) (frd(i,j,k), j = 1, nmarch)
```

13. fccrd1, fccrd2

```
      do 6 k = 1, 2
         do 5 i = 1, nmarch
5           read(50,3) (fccrd1(i,j,k), j = 1, nroad)
         read(50,3) (fccrd2([irsoff](k) + i), i = 1, nrs(k))
6        continue
```

14. vair

```
      do 5 j = 1, 2,
5        read(50,3) (vair([irsoff](j) + i), i = 1, nrs(j))
```

15. vsea

```
      do 5 j = 1, 2
5        read(50,3) (vsea([irsoff](j) + i), i = 1, nrs(j))
```

16. ifsea

```
      read(50,4) (ifsea(i), i = 1, nenv)
4     format (8l10)
```

17. ldcap, ldreq

```
      do 10 k = 1, 2
         do 8 i = 1, nrs(k)
8           read(50,3) (ldcap(i,j,k), j = 1, ntc)
         do 9 i = 1, nrs(k)
9        read(50,3) (ldreq(i,j,k), j = 1, ntc)
10    continue
```

18. ssreqm

```
    do 5 k = 1, 2
       do 5 j = 1, nrs(k)
5         read(50,3) (ssreqm(i,j,k), i = 1, nss(k))
```

19. barrr

```
    read(50,4) (barr(i), i = 1, nbar)
4   format (8l10)
```

20. bardly

```
    do 5 i = 1, nutype
       do 5 j = 1, nmarch
5         read(50,3) (bardly(i,j,k), k = 1, nbar)
```

Section: ssuse

1. ssvncr

```
    do 5 i = 1, nss(1)
       do 5 j = 1, nrs(1)
5         read(50,3) (ssvncr(i,j,k), k = 1, 3)
```

2. ssvncb

```
    do 5 i = 1, nss(2)
       do 5 j = 1, nrs(2)
5         read(50,3) (ssvncb(i,j,k), k = 1, 3)
```

3. mapps, ssvact, ssvres

```
      do 10 side = 1, 2
1     read(50,2) i, k
      if (i.le.0) go to 10
      mapps(side,[kpost](i)) = k
      read(50,4) old
4     format (l10)
      if (old) go to 1
      do 6 i = 1, nss(side)
      read(50,3) (ssvact(i,j,k), j = 1, nrs(side))
      read(50,3) (ssvres(i,j,k), j = 1, nrs(side))
6     continue
      go to 1
10    continue
```

Section: cmbt

1. poff

```
    k = 0
    do 5 i = 10, 9 + npost(1)
```

```
          k = k + 1
          poff(i) = k
    5     continue
       do 6 i = 20, 19 + npost(2)
          k = k + 1
          poff(i) = k
    6     continue
       do 7 i = 40, 39 + npost(4)
          k = k + 1
          poff(i) = k
    7     continue
       do 8 i = 10 + npost(1), 19
    8     poff(i) = poff(10)
       do 9 i = 20 + npost(2), 29
    9     poff(i) = poff(20)
       do 10 i = 40 + npost(4), 49
   10     poff(i) = poff(40)
   11  read(50,2) i, k
       if (i.le.0) go to 12
       if (i.le.29) poff(i) = max(k,poff(10))
       if (i.ge.40) poff(i) = max(k,poff(40))
       go to 11
   12  continue
```

2. stdtgt

```
       do 5 j = 1, 2
    5     read(50,3) (stdtgt(i,j), i = 1, nrs(j))
```

3. aggatk

```
       do 5 k = 1, 2
          do 5 i = 1, nggwep(k)
    5        read(50,3) (aggatk(i,j,k), j = 1, nmat(3-k))
```

4. aggdef

```
       do 5 k = 1, 2
          do 5 i = 1, nggwep(k)
    5        read(50,3) (aggdef(i,j,k), j = 1, nmat(3-k))
```

5. katk

```
       do 5 k = 1, 2
          do 5 = 1, nggwep(k)
    5        read(50,3) (katk(i,j,k), j = 1, nmat(3-k))
```

6. kdef

```
       do 5 k = 1, 2
          do 5 i = 1, nggwep(k)
    5        read(50,3) (kdef(i,j,k), j = 1, nmat(3-k))
```

7. fckar

```
      kap = 0
      do 5 i = 40, 49
 5       kap = max(poff(i), kap)
      do 6 j = 1, 2
         do 6 i = 1, nggwep(j)
 6          read(50,3) (fckar(i,j,k), k = 1, kap)
```

8. fckdr

```
      kdp = 0
      do 5 i = 10, 29
 5       kdp = max(poff(i), kdp)
      do 6 j = 1, 2
         do 6 i = 1, nggwep(j)
 6          read(50,3) (fckdr(i,j,k), k = 1, kdp)
```

9. fckac

```
      do 5 j = 1, 2
         do 5 i = 1, nmat(j)
 5          read(50,3) (fckac(i,j,k), k = 1, kdp)
```

10. fckdc

```
      do 5 j = 1, 2
         do 5 i = 1, nmat(j)
 5          read(50,3) (fckdc(i,j,k), k = 1, kap)
```

11. fckare

```
      do 5 j = 1, 2
         do 5 i = 1, nggwep(j)
 5          read(50,3) (fckare(i,j,k), k = 1, nenv)
```

12. fckdre

```
      do 5 j = 1, 2
         do 5 i = 1, nggwep(j)
 5          read(50,3) (fckdre(i,j,k), k = 1, nenv)
```

13. fckace

```
      do 5 j = 1, 2
         do 5 i = 1, nmat(j)
 5          read(50,3) (fckace(i,j,k), k = 1, nenv)
```

14. fckdce

```
    do 5 j = 1, 2
       do 5 i = 1, nmat(j)
 5        read(50,3) (fckdce(i,j,k), k = 1, nenv)
```

15. fckarb

```
    do 5 j = 1, 2
       do 5 i = 1, nggwep(j)
 5        read(50,3) (fckarb(i,j,k), k = 1, nbar)
```

16. ifrep

```
    read(50,4) ifrep
  4 format (l10)
```

17. frdmg

```
    if (.not.ifrep) go to 6
    do 5 side = 1, 2
       k0 = 0
       if (side.eq.2)k0 = nequip(1)
       do 4 k = 1, nequip(side)
 4        read(50,3) (frdmg(i,j,k0+k), j = 1, nggwep - side))
 5     continue
  6 continue
```

18. dpersr

```
    do 5 i = 1, npers(1)
       do 5 j = 1, nggwep(2)
 5        read(50,3) (dpersr(i,j,k), k = 1, nmat(1))
```

19. dpersb

```
    do 5 i = 1, npers(2)
       do 5 j = 1, nggwep(1)
 5        read(50,3) (dpersb(i,j,k), k = 1, nmat(2))
```

20. td

```
    read(50,3) (td(i), i = 1, nenv)
```

21. febab

```
    read(50,3) (febab(i), i = 1, nbarty)
```

22. febad

```
    read(50,3) febad
```

22. vanish

```
      read(50,3) (vanish(i), i = 1, nutype)
```

23. quit

```
      do 5 j = 1, 2
    5    read(50,3) (quit(i,j), i = 1, npost(4))
```

24. stdfor

```
      do 5 i = 1, 2
    5    read(50,3) (stdfor(i,j), j = 1, nrs(i))
```

Section: frdv

1. frdval.f0atk, frdval.fatk

```
      do 5 i = 40, 39+npost(4)
    5    read(50,3) frdval.f0atk(poff(i)),
                    (frdval.fatk(poff(i),j), j = 1, 7)
```

2. frdval.f0def, frdval.fdef

```
      do 5 i = 10, 9+npost(1)
    5    read(50,3) frdval.f0def(poff(i)),
                    (frdval.fdef(poff(i),j), j = 1, 7)
      do 6 i = 20, 19+npost(2)
    6    read(50,3) frdval.f0def(poff(i)),
                    (frdval.fdef(poff(i),j), j = 1, 7)
```

3. frdval.x

```
      read(50,3) (frdval.x(i), i = 1, 7)
    4 format (10x, 7f10.0)
```

Section: vfeba

1. vfeba.f0, vfeba.f

```
      vfeba.npa = 6
    1 read(50,2) i, j
      if (i.le.0) go to 5
      read(50,3) vfeba.f0(poff(i), poff(j)),
                 (vfeba,f(poff(i), poff(j),k), k = 1, 7)
      go to 1
    5 read(50,2) i, j
      if (i.le.0) go to 6
      read(50,3) vfeba.f0(vfeba.npa+poff(i), poff(j)),
                 (vfeba.f(vfeba.npa+poff(i), poff(j),k), k = 1, 7)
      go to 5
    6 continue
```

2. vfeba.fr

```
      read(50,4) (vfeba.fr(i), i = 1, 7)
    4 format (10x, 7f10.0)
```

Section: frinv

1. ppoh

```
      do 5 j = 1, nutype
    5    read(50,3) (ppoh(i,j), i = 1, npers(flag(j)))
```

2. spdd

```
      do 5 j = 1, nrs(1)
    5    read(50,3) (spdd(i,j), i = 1, nsp(1))
```

3. frinv.f0, frinv.f

```
      do 5 j = 1, nmat(1)
         do 5 i = 1, nsp(1)
    5       read(50,3) frinv.f0(i,j)
                       (frinv.f(i,j,k), k = 1, 6)
```

4. frinv.x(1,*)

```
      read(50,4) (frinv.x(1,i), i = 1, 6)
    4 format (10x, 7f10.0)
```

5. spdd

```
      do 5 j = 1, nrs(2)
    5    read(50,3) (spdd(i,nrs(1)+j), i = 1, nsp(2))
```

6. frinv.f0, frinv.f

```
      do 5 j = 1, nmat(2)
         do 5 i = 1, nsp(2)
    5       read(50,3) (frinv.f0(i,nrs(1)+j),
                       (frinv.f(i,nrs(1)+j,k), k = 1, 6))
```

7. frinv.x(2,*)

```
      read(50,4) (frinv.x(2,i), i = 1, 6)
    4 format (10x, 7f10.0)
```

8. pg, prot

```
      do 5 k = 1, 2
         read(50,3) (pg(i,k), i = 1, nequip(k))
         do 5 i = 1, nequip(k)
```

```
5       read(50,3) (prot(i,j,k), j = 1, nggwep(k))
6    continue
```

Section: freff

1. freff.f0, freff.f, freff.x

```
     do 5 i = 1, 2
        read(50,3) (freff.f0(i), (freff.f(i,j), j = 1, 7))
        read(50,4) (freff.x(i,j), j = 1, 7)
4       format (10x, 7f10.0)
5       continue
```

Section: prep

1. prep.f, prep.x

```
     do 6 j = 1, 2
        do 5 i = 1, nmat(j)
5          read(50,3) (prep.f(i,j,k), k = 1, 7)
           continue
        read(50,3) (prep.x(j,k), k = 1, 7)
6       continue
```

Section: sf

1. ifsa

```
        read(50,4) (ifsa(10,i), i = 1,2)
4       format (2l10)
```

2. rngfc, range

```
        do 6 j = 1,2
           if (.not. ifsa(10,j)) go to 6
           do 5 i = 1, nggwep(j)
5           read(50,3) (range(j,k), k = 1,6)
           read(50,3) (range(j,k), k = 1,6)
6          continue
```

3. postfc

```
        do 6 j = 1,2
           if (.not. ifsa(10,j)) to to 6
           do 5 i = 1, nggwep(j)
5           read(50,3) (postfc(j,k), k = 1, npost(1))
6          continue
```

4. spreqf

```
      do 6 k = 1,2
         if (.not. ifsa(10,k) .or. nsp(k) = 0) go to 6
         do 5 j = 1, nggwep(k)
5        read(50,3) (spreqf(i,j,k), i = 1, nap(k))
6        continue
```

Section: rep

1. repdd

```
      if (.not. ifrep) go to 6
      do 5 k = 1, 2
         do 4 i = 1, nequip(k)
4           read(50,3) (repdd(i,j,k), j = 1, 2)
5        continue
6     continue
```

2. repcu

```
      if (ifrep) read(50,3) (repcu(i), i = 1, nutype
```

3. repcs

```
      if (ifrep) read(50,3) (repcs(i), i = 1, 2)
```

Section: air

1. nactyp, nagwep

```
      do 5 i = 1, 2
5        read(50,2) nactyp(i), nagwep(i)
```

2. kag

```
      do 5 k = 1, 2
         do 5 i = 1, nagwep(k)
5           read(50,3) (kag(i,j,l), j = 1, nmat(3-k))
```

3. fcagrp

```
      do 5 j = 1, 2
         do 5 i = 1, nagwep(j)
5           read(50,3) (fcagrp(i,j,k), k = 1, 4)
```

4. fcagcp

```
      do 5 j = 1, 2
         do 5 i = 1, nmat(j)
5           read(50,3) (fcagcp(i,j,k), k = 1, 4)
```

5. fcagre

```
      do 5 j = 1, 2
         do 5 i = 1, nagwep(j)
5           read(50,3) (fcagre(i,j,k), k = 1, nenv)
```

6. fcagce

```
      do 5 j = 1, 2
         do 5 i = 1, nmat(j)
5           read(50,3) (fcagce(i,j,k), k = 1, nenv)
```

7. dgpred

```
      do 5 k = 1, nmat(1)
         do 5 i = 1, npers(1)
5           read(50,3) (dgpred(i,j,k), j = 1, nagwep(2))
```

8. dgpblu

```
      do 5 k = 1, nmat(2)
         do 5 i = 1, npers(2)
5           read(50,3) (dgpblu(i,j,k), j = 1, nagwep(1))
```

9. agload

```
      do 5 k = 1, 2
         do 5 i = 1, nactyp(k)
5           read(50,3) (agload(i,j,k), j = 1, nagwep(k))
```

10. aagatk(*,*,1)

```
      do 5 i = 1, nagwep(1)
5        read(50,3) (aagatk(i,j,1), j = 1, nmat(2))
```

11. aagdef(*,*,1)

```
      do 5 i = 1, nagwep(1)
5        read(50,3) (aagdef(i,j,1), j = 1, nmat(2))
```

12. aagatk(*,*,2)

```
      do 5 i = 1, nagwep(2)
5        read(50,3) (aagatk(i,j,2), j = 1, nmat(1))
```

13. aagdef(*,*,2)

```
      do 5 i = 1, nagwep(2)
    5    read(50,3) (aagdef(i,j,2), j = 1, nmat(1))
```

14. aagred

```
      do 5 i = 1, nagwep(1)
         do 5 j = 1, nmat(2)
    5       read(50,3) (aagred(i,j,k), k = 1, 3)
```

15. aagblu

```
      do 5 i = 1, nagwep(2)
         do 5 j = 1, nmat(1)
    5       read(50,3) (aagblu(i,j,k), k = 1, 3)
```

Section: haven

1. haven.zoc

```
      read(50,4) haven.zoc
    4 format (l10)
```

2. haven.love

```
      do 5 i = 1, 2
    5    read(50,3) (haven.love(i,j), j = 1, 6)
```

Section: locma

1. nbar0

```
      read(50,2) nbar0
```

2. ifsa, ifrole

```
      do 5 j = 1, 2
      read(50,4) (ifsa(i,j), i = 1, 7)
    5 read(50,4) (ifrole(i,j), i = 1, 3)
    4 format (8l10)
```

3. bistep, bifx, biperm

```
      if (.not. ifsa(1,1) .and..not. ifsa(1,2) .and.
          .not. ifrole(1,1) .and..not. ifrole(1,2)) go to 10
      read(50,3) (bistep(i), i = 1, nbar0)
      read(50,2) (bifx(i), i = 1, nbar0)
      read(50,4) (biperm(i), i = 1, nbar0)
    4 format (8l10)
   10 continue
```

4. locmop

```
      read(50,2) (locmop(i), i = 1, 7)
```

5. cecap

```
      do 6 k = 1, 2
         do 5 j = 1, 7
            if (.not. ifsa(j,k)) go to 6
5           read(50,3) (cecap(irsoff(k)+i,j), i = 1, nrs(k))
6           continue
```

6. desreq, deseps

```
      if (.not. ifsa(2,1) .and. .not. ifsa(2,2) .and.
         .not. ifsa(3,1) .and. .not. ifsa(3,2) go to 10
      read(50,3) (desreq(i), i = 1, nroad + nrail)
      read(50,3) (deseps(i), i = 1, nroad + nrail)
   10 continue
```

7. bmstep, bmfx

```
      if (.not. ifsa(7,1) .and..not. ifsa(7,2)) go to 10
      read(50,3) (bmstep(i), i = 1, nbar0)
      read(50,2) (bmfx(i), i = 1, nbar0)
   10 continue
```

8. repreq

```
      if (ifsa(5,1) .or. ifsa(5,2) .or.
         ifsa(6,1) .or. ifsa(6,2))
      read(50,3) (repreq(i), i = 1, nroad + nrail)
```

9. ssreqe

```
      do 8 side = 1, 2
         do 8 k = 1, 7
            if (.not. ifsa(k,side)) go to 8
            do 5 j = 1, nrs(side)
5              read(50,3) (ssreqe(i,j,k,side), i = 1, nss(side))
8        continue
```

10. cecapa

```
      do 10 k = 1, 2
         if (nactyp(k) .eq. 0 .or.
             nagwep(k) .eq. 0) go to 10
         i0 = 0
         if (k .eq. 2) i0 = nagwep(1)
         do 5 j = 1, 3
            if (.not. ifrole(j,k)) go to 5
            read(50,3) (cecapa(i0+i,k), i = 1, nagwep(k))
    5       continue
   10    continue
```

11. desrqa, desepa

```
     if (.not. ifrole(2,1) .and. .not. ifrole(2,2) .and.
         .not. ifrole(3,1) .and. .not. ifrole(3,2)) go to 10
     read(50,3) (desrqa(i), i = 1, nroad + nrail)
     read(50,3) (desepa(i), i = 1, nroad + nrail)
```

10.2 FILE 60

The following data are read from file 60 at the start of each cycle:

1. mapter

```
    1 read(60,2,end=5) i,j
      if (i.le.0) go to 5
      mapter(i) = j
      go to 1
    5 continue
```

2. maprd

```
    1 read(60,2,end=5) i,j
      if (i.le.0) go to 5
      maprd(i) = j
      go to 1
    5 continue
```

3. maprr

```
    1 read(60,2,end=5) i,j
      if (i.le.0) go to 5
      maprr(i) = j
      go to 1
    5 continue
```

4. mapmb

```
      1 read(60,2,end=5) i,j
        if (i.le.0) go to 5
        mapmb(i) = j
        go to 1
      5 continue
```

5. mapab

```
      1 read(60,2,end=5) i,j
        if (i.le.0) go to 5
        mapab(i) = j
        go to 1
      5 continue
```

IDAHEX sets

```
mapter(i) = i for every i,
 maprd(i) = i for every i,
 maprr(i) = i for every i,
 mapmb(i) = i for every i,
 mapab(i) = i for every i,
```

before the start of a game. At the start of each cycle, including the first, it reads file 60 for redefinitions of mapter, maprd, maprr, mapmb, and mapab.

12. GLOSSARY

This section contains an alphabetical glossary of variables and functions mentioned in this volume. For each variable such that array dimensions or consistency of the game design data implies a finite upper or lower bound on the variable's value, that bound is given; "UB" and "LB" are abbreviations of "upper bound" and lower bound". A variable may attain its upper or lower bound. Variables whose names end in "max" or "mx" define upper bounds imposed by the sizes of array dimensions. Their values are set in the IDAHEX main program, cgcm.

For the sake of succinctness, the following conventions apply:

1. "Type i artery" means "type i road system" if $1 \leq i \leq nroad$, and "type i-*nroad* rail system" if $i > nroad$.

2. "Type i resources", without any reference to their side, means "side 1 type i resources" if $i \leq nrs(1)$, and "side 2 type i-nrs(1) resources" if $i > nrs(1)$.

3. "Type i air-to-ground weapons", without any reference to their side, means "side 1 type i air-to-ground weapons" if $i \leq nagwep(1)$, and "side 2 type i-*nagwep*(1) air-to-ground weapons" if $i > nagwep(1)$.

Name	Description	Type
aagatk(i,j,k)	fraction of fire of side k air-to-ground weapons of type i allocated to enemy materiel of type j when enemy materiel belongs to engaged battle unit in attack posture LB = 0	real
aagblu(i,j,k)	fraction of fire of Blue air-to-ground weapons of type i allocated to enemy materiel of type j when enemy materiel belongs to unengaged battle unit in posture class k LB = 0	real
aagdef(i,j,k)	fraction of fire of side k air-to-ground weapons of type i allocated to enemy materiel of type j when enemy materiel belongs to engaged battle unit in hold or disengagement posture LB = 0	real
aagred(i,j,k)	fraction of fire of Red air-to-ground weapons of type i allocated to enemy materiel of type j when enemy materiel belongs to unengaged battle unit in posture class k LB = 0	real
abname(i,*)	description of type i movement barrier	character
aggatk(i,j,k)	fraction of fire of side k ground-to-ground weapons of type i allocated to enemy materiel of type j if side k is engagement attacker LB = 0	real
aggdef(i,j,k)	fraction of fire of side k ground-to-ground weapons of type i allocated to enemy materiel of type j if side k is engagement defender LB = 0	real

Name	Description	Type
agload(i,j,k)	notional load of side k air-to-ground weapons of type j on side k aircraft of type i LB = 0	real
aisize(i,j)	area of area of responsibility of a unit of type i in posture class j if its resources coincide with *toe*(i,*) LB = 0	real
bardly(i,j,k)	barrier delay for a unit of type i in j-th movement posture crossing a barrier of type k LB = 0	real
[barrier](i,j)	type of barrier between cell i and cell j (0 signifies no barrier); undefined unless cells are adjacent LB = 0, UB = *nbar*	integer
barrr(i)	true if and only if type i barrier blocks rail traffic	logical
[basic_barrier](i,j)	basic type of barrier between cell i and cell j (0 signifies no barrier); undefined unless cells are adjacent LB = 0, UB = *nbar0*	integer
[*basic_barrier*](i,j)	basic type of barrier between cell i and cell j at start of game (0 signifies no barrier); undefined unless cells are adjacent LB = 0, UB = *nbar0*	integer
[*basic_ rail*](i,j)	basic type of rail link between cell i and cell j (0 signifies no railroad); undefined unless cells are adjacent LB = 9, UB = *nrraw*	integer
[*basic_road*](i,j)	basic type of road link between cell i and cell j (0 signifies no road); undefined unless cells are adjacent LB = 0, UB = *nrdraw*	integer

Name	Description	Type
bifx(i)	basic barrier type that results when barrier of basic type i is subjected to level *bistep*(i) of barrier intensification LB = 1, UB = *nbar0*	integer
biperm(i)	.true. if and only if type i barrier resulting from barrier intensification becomes new basic barrier type	logical
bistep(i)	level of barrier intensification effort needed to transform barrier of basic type i into another type LB > 0	real
bmfx(i)	basic barrier type that results when barrier of basic type i gets barrier mitigation effort equal to *bmstep*(i) LB = 1, UB = *nbar0*	integer
bmstep(i)	level of barrier mitigation effort needed to transform barrier of basic type i into another basic type LB > 0	real
bname0(i,*)	description of basic barrier type i	character
buloc(i)	location of unit i	integer
buloc(i)	location of unit i (a cell number) at start of game LB = 1, UB = *ncells*	integer
buname(i,*)	name of unit i	character
bupost(i)	posture of unit i	integer
bupost(i)	posture of unit i at start of game LB = 0, UB = 19	integer
butype(i)	type of unit i LB = 1, UB = *nutype*	integer

Name	Description	Type
cecap(i,j)	capability of type i resources in type j special activity LB = 0	real
cecapa(i,k)	capability of type i air-to-ground weapons in type k strike role LB = 0	real
celoc1(i)	lower-numbered cell incident to LOC affected by LOC effect	integer
celoc2(i)	higher-numbered cell incident to LOC affected by LOC effect	integer
celvl(i)	intensity of LOC effect	real
cetype(i)	type of LOC effect	integer
delta	length of time a unit must be in movement posture before arrival of enemy unit at its location (point of origin) to avoid reversion to disengagement posture LB = 0	real
depth	distance from center of any cell to center of adjacent cell LB > 0	real
desepa(i)	length of type i artery that air strike must destroy for damage to be considered significant LB = 0	real
deseps(i)	length of type i artery that ground forces must destroy for damage to be considered significant LB = 0	real
desreq(i)	ground resources' effort needed to destroy unit-length of type i artery LB > 0	real

Name	Description	Type
desrqa(i)	effort needed to destroy unit-length of type i artery by air strikes LB > 0	real
dgpblu(i,j,k)	loss of Blue personnel of type i associated with destruction of unit-quantity of Blue materiel of type k by Red air-to-ground weapons of type j LB = 0	real
dgpred(i,j,k)	loss of Red personnel of type i associated with destruction of unit-quantity of Red materiel of type k by Blue air-to-ground weapons of type j LB = 0	real
dirab(i,j)	effective attack barrier type for attack in direction j if preliminary estimate of attack barrier type is i LB = 0, UB = *nab*	integer
diseng(i)	minimum time required for a type i unit to disengage LB = 0	real
dpersb(i,j,k)	loss of Blue personnel of type i associated with destruction of a unit-quantity of Blue materiel of type k by Red ground-to-ground weapons of type j LB = 0	real
dpersr(i,j,k)	loss of Red personnel of type i associated with destruction of a unit-quantity of Red materiel of type k by Blue ground-to-ground weapons of type j LB = 0	real
elude(i,j)	factor applied to movement delay to find additional disengagement delay imposed on side j type i resources disengaging without a rearguard LB = 0	real

Name	Description	Type
ename(i,*)	description of environment type i	character
ename0(i,*)	description of basic environment type i	character
[environment](i)	type of environment in cell i LB = 0, UB = *nenv*	integer
fcagce(i,j,k)	factor applied to *kag*(*,i,3-j) if target battle unit is in environment k LB = 0	real
fcagcp(i,j,k)	factor applied to *kag*(*,i,3-j) if target battle unit is in posture class k LB = 0	real
fcagre(i,j,k)	factor applied to *kag*(i,*,j) if target battle unit is in environment k LB = 0	real
fcagrp(i,j,k)	factor applied to (i,*,j) if target battle unit is in posture class k LB = 0	real
fcc(i,j)	adjustment factor applied to cross-country movement rate of side j resources in movement posture i LB = 0	real
fccrd1(i,j,k)	factor indicating propensity of side k task force in movement posture i to travel off-road when road link type is j LB = 0, UB = 1	real
fccrd2(i)	propensity of type i resources to travel off-road LB = 0, UB = 1	real
fckac(i,j,k)	factor applied to katk(*,i,3-j) if materiel belongs to battle unit in posture p (10 ≤ p ≤ 29); k = *poff*(p) LB = 0	real

Name	Description	Type
[fckac](i,j,k)	factor applied to katk(*,i,3-j) if materiel belongs to battle unit in posture k; it equals *fckac*(i,j,*poff*(k)) LB = 0	real
fck[illegible](i,j,k)	factor applied to katk(*,i,3-j) if environment in engagement cell is type k LB = 0	real
fckar(i,j,k)	factor applied to katk(i,*,j) if weapon belongs to battle unit in posture p (40 ≤ p ≤ 49); k = *poff*(p) LB = 0	real
[fckar](i,j,k)	factor applied to katk(i,*,j) if weapon belongs to battle unit in posture k; it equals *fckar*(i,j,*poff*(k)) LB = 0	real
fckarb(i,j,k)	factor applied to katk(i,*,j) if weapon i belongs to battle unit attacking across barrier of type k and engagement feba ≤ *febab*(k)/[illegible]*epth* LB = 0	real
fckare(i,j,k)	factor applied to katk(i,*,j) if environment in engagement cell is type k LB = 0	real
fckdc(i,j,k)	factor applied to kdef(*,i,3-j) if materiel belongs to battle unit in posture p (10 ≤ p ≤ 29); k = *poff*(p) LB = 0	real
[fckdc](i,j,k)	factor applied to kdef(*,i,3-j) if materiel belongs to battle unit in posture k; it equals *fckdc*(i,j,*poff*(k)) LB = 0	real

Name	Description	Type
fckdce(i,j,k)	factor applied to kdef(*,i,3-j) if environment in engagement cell is type k LB = 0	real
fckdr(i,j,k)	factor applied to kdef(i,*,j) if weapon belongs to battle unit in posture p (10 ≤ p ≤ 29); k = *poff*(p) LB = 0	real
[fckdr](i,j,k)	factor applied to kdef(i,*,j) if weapon belongs to battle unit in posture k; it equals *fckdr*(i,j,*poff*(k)) LB = 0	real
fckdre(i,j,k)	factor applied to kdef(i,*,j) if environment in engagement cell is type k LB = 0	real
febab(i)	depth of attacker penetration of defender's cell at which effect of type i barrier ceases LB = 0, UB = *depth*	real
febad	degree of attacker penetration of defender's cell at which defenders must disengage and retreat to another cell LB = 0, UB < 1	real
fercl(i)	load class of ferries in force moving in posture 29 + i LB = 1	integer
flag(i)	side to which unit of type i belongs	integer
[floor](a)	largest integer ≤ a	integer
frd(i,j,k)	adjustment factor applied to road movement rate of side k type i equipment in movement posture j LB = 0	real

Name	Description	Type
frdmg(i,j,k)	fraction of a side s unit's disabled type ℓ equipment that goes to repair class i if disabled by enemy type i weapons, where k = ℓ if s = 1 and k = nequip(1) + ℓ if s = 2 LB = 0, UB = 1	real
frdval.fatk(i,j)	fraction of value lost by attackers in 1 unit of time when attacker-to-defender force ratio is *frdval.x*(j) and attacker's posture is p; i = *poff*(p) LB = 0, UB = 1	real
frdval.fdef(i,j)	fraction of value lost by defender in 1 unit of time when attacker-to-defender force ratio is *frdval.x*(j) and defender's posture is p; i = *poff*(p) LB = 0, UB = 1	real
frdval.f0atk(i)	fraction of value lost by attacker in 1 unit of time when attacker-to-defender force ratio is 0 and attacker's posture is p; i = *poff*(p) LB = 0, UB = 1	real
frdval.f0def(i)	fraction of value lost by defenders in 1 unit of time when attacker-to-defender force ratio is 0 and defender's posture is p; i = *poff*(p) LB = 0	real
frdval.x(i)	ordinate corresponding to *frdval.fatk*(j,i) and *frdval.fdef*(j,i) for any j LB = 0	real
freff.f(i,j)	fractional effectiveness in combat of one or more side i battle units located in same cell if total area of their areas of responsibility divided by area of cell equals *freff.x*(i,j) LB = 0	real

Name	Description	Type
freff.f0(i)	fractional effectiveness in combat of one or more side i battle units located in same cell if total area of their zone of responsibility is 0 LB = 0	real
freff.x(i,j)	ordinate corresponding to *freff.f*(i,j) LB = 0	real
frinv.f(i,j,k)	fraction of type r resources available for combat in side s battle unit if available quantity of type i support resources divided by demand for type i support resources equals *frinv.x*(s,k); j = r if s = 1, j = nrs(1)+r if s = 2 LB = 0	real
frinv.f0(i,j)	fraction of type r resources available for combat in side s battle unit if available quantity of type i support resources divided by demand for type i support resources equals 0; j = r if s = 1, j = nrs(1)+r if s = 2 LB = 0	real
frinv.x(i,j)	ordinate corresponding to frinv.f(k,ℓ,j) for any (k,ℓ) associated with side i LB = 0	real
haven.love(i,j)	measure of preference of side i for retreating in direction of j-th rim cell ($1 \leq j \leq 6$)	real
haven.zoc	truth value of "attacker's zone of control extends into adjacent cells"	logical
iars(i,j)	absolute index of i-th resource on list of resources in a unit of type j LB = 0, UB = nrs(*flag*(j))	integer

Name	Description	Type
*iblu*l	index of number of lowest-numbered Blue unit LB = 2, UB = nbumax	integer
ifrep	true if and only if maintenance is played	logical
ifrole(i,j)	true if and only if side j type i air strike role is played	logical
ifsa(i,j)	true if and only if side j type i special activity is played	logical
ifsea(i)	true if and only if type i environment is water	logical
imclas(i,j)	lowest load class of independently moving resources in battle unit of type j in movement posture i LB = 1, UB = *nlc*	integer
iprint	level of detail in output LB = 0	integer
irsoff(s)	0 if s = 1, nrs(1) if s = 2	integer
itrfp	index of transfer posture ($10 \leq i \leq 19$) LB = 10, UB = 9 + *npost*(1)	integer
ilrep2	initial point for recording repair class 2 information in repbu and reppool	integer
kag(i,j,k)	amount of enemy materiel of type j destroyed by a single side k air-to-ground weapon of type i if all of the air-to-ground weapon's fire is allocated to enemy materiel of type j LB = 0	real
kap	max [*poff*(i); $40 \leq i \leq 49$]	integer

Name	Description	Type
katk(i,j,k)	amount of enemy materiel of type j destroyed in 1 unit of time by a single side k ground-to-ground weapon of type i if the weapon allocates all its fire to enemy materiel of type j; side k is the attacker in the engagement LB = 0	real
katk(i,j,k)	*tframe* * *katk*(i,j,k)	real
kdef(i,j,k)	amount of enemy materiel of type i destroyed in 1 unit of time by a single side k ground-to-ground weapon of type i if the weapon allocates all its fire to enemy materiel of type j; side k is the defender in the engagement LB = 0	real
kdef(i,j,k)	*tframe* * *kdef*(i,j,k)	real
kdp	max [*poff*(i); $10 \leq i \leq 29$]	real
[kpost](p)	p - 9 if $p < 40$, p - 19 if $p \geq 40$	integer
ldcap(i,j,k)	load capacity in load class j of side k type i resources	real
ldclas(i,j)	load class of side j type i resources LB = 0, UB = *ntcmax*	integer
ldreq(i,j,k)	burden of a single side k type i resource relative to load class j LB = 0	real
mapab(i)	attack barrier type if basic barrier type is i LB = 0, UB = *nab*	integer
mapmb(i)	movement barrier type if basic barrier type is i LB = 0, UB = *nmb*	integer

Name	Description	Type
mapps(i,j)	pointer used to reference data on supplies consumption in engaged side i battle unit in posture p, where j = [kpost](p) (see *ssvact* and *ssvres*) LB = 1, UB = ssuse,npsmax	integer
maprd(i)	road type if basic road type is i LB = 0, UB = *nroad*	integer
maprr(i)	railroad type if basic rail type is i LB = 0, UB = *nrail*	integer
mapter(i)	environment type if basic environment type is i LB = 1, UB = *nenv*	integer
mbname(i,*)	description of type i movement barrier	character
mode(i,s)	mode of self-transport of side s type i resources: 1 if road/cross-country or no self-transport capability; 2 if air; 3 if rail; 4 if sea LB = 1, UB = 4	integer
nab	maximum value of (atkbar)(i,j) LB = 0, UB = ncbmax	integer
nactyp(i)	number of side i aircraft types LB = 0, UB = nacmax	integer
nagwep(i)	number of side i air-to-ground weapon types LB = 0, UB = nagwmx	integer
nbar0	maximum basic barrier type that can occur LB = 0, UB = nbarraw	integer
ncells	number of cells (largest identification number of any cell) in area of war LB = 1, UB = ncelmx	integer
nenv	maximum value of [environment](i) LB = 1, UB = nenvmx	integer

Name	Description	Type
nequip(i)	number of types of side i equipment (nwep(i) + *ntrpt*(i)) LB = 1	integer
ngawep(i)	number of types of side i ground-to-air weapons LB = 0, UB = neqmax	integer
nggwep(i)	number of types of side i ground-to-ground weapons LB = 1, UB = nggwmx	integer
nmarch	largest k such that a task force moving in posture 29 + k may have an independently moving resource traveling by road or cross-country LB = 1, UB = nmpmax	integer
nmat(i)	number of types of side i materiel (nwep(i) + *ntrpt*(i) + *nss*(i)) LB = 1, UB = nmatmx	integer
nmb	maximum value of movebar(i,j) LB = 0, UB = nmbmax	integer
nmpmax	nmpmax + 29 = highest permitted index of posture implying cross-country or road movement	integer
nnsyl	number of computer double-words occupied by name of any environment, road, rail, or barrier type	integer
npers(i)	number of types of side i personnel LB = 0, UB = npermx	integer
npost(i)	number of postures in posture class i ($1 \le i \le 4$) LB = 1, UB = 10	integer
nprint	number of output devices (printer & terminals) to be used LB = 1, UB = 3	integer
nrail	maximum value of [rail](i,j) LB = 0, UB = nrrmax	integer

Name	Description	Type
*nrank*1	number of cells in first row of area of war LB = 2, UB = *ncells*	integer
nroad	maximum value of [road](i,j) LB = 0, UB = nrdmax	integer
nrs(i)	number of types of side i resources LB = 1, UB = nrsmax	integer
nrst(i)	number of types of resources that a unit of type i may have LB = 1, UB = nrs(*flag*(i))	integer
nsp(i)	number of types of side i support resources (*nss*(i) + *npers*(i))	integer
nss(i)	number of types of side i supplies LB = 0, UB = nssmax	integer
nsyl	number of computer double-words occupied by name of any unit LB = i, UB = 2	integer
nlc	max $\{ldclas(i,s);\ 1 \le s \le 2,\ 1 \le i \le nrs(s)\}$ LB = 1, UB = nlcmax	integer
ntrpt(i)	number of types of side i transport vehicles LB = 0, UB = ntrnmx	integer
nutype	number of types of units LB = 1, UB = nutymx	integer
nwep(i)	number of types of side i weapons (*nggwep*(i) + *ngawep*(i)) LB = 1, UB = nwepmx	integer
[owner](i)	1 if Red owns cell i, 2 if Blue	integer
[*owner*](i)	side that owns cell i at start of game	integer
pg(i,j)	protection group to which side j resources of type i belongs LB = 1	integer

Name	Description	Type
pmapdn(i)	first posture a unit enters when it transitions from posture class i to a lower posture class	integer
pmapup(i)	first posture a unit enters when it transitions from posture i to a higher posture class	integer
poff(i)	offset pointer used to reference ground combat data for a unit in posture i (see *fckar, fcka, frdval.fatk, vfeba.f*)	integer
postfc(i,j,k)	factor reducing effective number of side j type i ground-to-ground weapons for supporting fire in unit in posture 9 + k LB = 0, UB = 1	real
ppoh(i,j)	number of overhead type i personnel in type j battle unit LB = 0	real
prep.f(i,j,k)	factor applied to katk(*,i,3-j) if defending unit, a member of side j, is credited with defense preparation time equal to *prep.x*(j,k) LB = 0	real
prep.x(i,j)	ordinate corresponding to *prep.f*(k,i,j) for any k	real
prot(i,j,k)	amount of side k equipment of type i that a side k ground-to-ground weapon of type j can protect LB = 0	real
ptran(i,j,k)	time required to transition from j-th posture in posture class i to k-th posture in posture class i LB = 0	real
quit(i,j)	combined ground-air force ratio below which side j attackers in posture i break off attack LB = 0	real

Name	Description	Type
[rail](i,J)	type of rail link between cell i and cell j (0 signifies no railroad); undefined unless cells are adjacent LB = 0, UB = *nrail*	integer
range(k,j)	ordinate corresponding to *rngfc*(*,i,j). LB = 0	real
repbu(i)	number of battle units to which reppool(*,i) corresponds, if any	integer
repcu(i)	repair capability in repair class 1 of type i battle unit LB = 0	real
repcs(i)	repair capability in repair class 2 of side i LB = 0	real
repdd(i,j,k)	effort needed to repair an item of side k type i equipment in repair class j LB > 0	real
reppool(i,j)	quantity of type i equipment in a particular repair class for unit repbu(j)	real
repreq(i)	effort needed to repair unit-length of type i artery LB > 0	real
rept(i)	time of first entry in reppool(*,ilrep2+i-1)	real
[resources](i,j)	quantity of resources of type j in unit i (classification of resources by type depends on unit's side)	real
[*resources*](i,j)	quantity of resources of type j in unit i at start of game LB = 0	real
rname(i,*)	description of road type i	character

Name	Description	Type
rname0(i,*)	description of basic road type i	character
[road](i,j)	type of road link between cell i and cell j (o signifies no road); undefined unless cells are adjacent LB = 0, UB = *nroad*	integer
rrname(i,*)	description of railroad type i	character
rrname0(i,*)	description of basic railroad type i	character
rsname(i,*,j)	description of side j resource type i	character
rsvala(i,j)	standard value of side j type i resources on attack	real
rsvald(i,j)	standard value of side j type i resources on defense	real
saloc1(i)	low-numbered cell where special activity occurs	integer
saloc2(i)	higher-numbered cell where activity occurs, or engagement location in case of support fire	integer
salv1(i)	intensity of special activity	integer
satype(i)	type of special activity	integer
saunit(i)	unit performing special activity	integer
spdd(i,j)	demand for type i support resources by a unit quantity of type r resources; j = r if resources belong to Red battle unit; j = mrs(1)+r is resources belong to Blue battle unit LB = 0	real
spreqf(i,j,k)	requirement of side k type j ground-to-ground weapons performing supporting fire for type i support resources LB = 0	real

Name	Description	Type
ssreqe(i,j,k,ℓ)	rate of consumption of type i supplies by side ℓ type j resources doing type k special activity	real
ssreqm(i,j,k)	quantity of side k supplies of type i required by a side k resource of type j in order to move LB = 0	real
ssvact(i,j,k)	quantity of type i supplies consumed in one unit of time by a type j resource actively involved in ground combat; supplies and resources belong to a battle unit from side s in posture p; k = *mapps*(s,[kpost](p)) LB = 0	real
ssvncb(i,j,k)	amount of type i supplies consumed in one unit of time by a type j resource in a Blue battle unit in posture class k; $1 \le k \le 3$ LB = 0	real
ssvncr(i,j,k)	amount of type i supplies consumed in one unit of time by a type j resource in a Red battle unit in posture class k; $1 \le k \le 3$ LB = 0	real
ssvres(i,j,k)	consumption of type i supplies in one unit of time by a type j resource not actively involved in combat but in an engaged battle unit; battle unit belongs to side s and is in posture p; k = *mapps*(s,[kpost](p)) LB = 0	real
stdfor(i,j)	quantity of resource j in a standard side i ground force LB = 0	real

Name	Description	Type
stdtgt(i,j)	quantity of resource i in a standard ground force opposed to side j LB = 0	real
t	current game time (t = *tinit* at start of game)	real
tcycle	length of cycle LB = *tpd*	real
td(i)	depth of defender's tactical zone when environment in engagement cell is type i LB = 0	real
tend	time at which game ends LB = *tinit*	real
tentry(i)	time at which unit i entered location and posture class it is in at start of game	real
tentry(i)	virtual time at which unit i entered its present posture class	real
[*terrain*](i)	basic environment (terrain) in cell i LB = 0, UB = nenvraw	integer
tframe	length of frame LB = 0, UB = *tpd*	real
tinit	time at start of game LB = 0	real
toe(i,j)	planned effective quantity of type j resources in a unit of type i LB = 0	real
tpd	length of period LB = *tframe*, UB = *tcycle*	real
vair(i)	rate at which type i resources can fly LB = 0	real

Name	Description	Type
vanish(i)	fraction of standard value at which battle unit of type i vanishes LB = 0, UB = 1	real
vcc(i,j)	rate at which type i resources can move themselves cross-country in type j environment LB = 0	real
vfeba.f(i,j,k)	signed distance of FEBA movement in 1 unit of time if attacker-to-defender force ratio is *vfeba.fr*(i), attacker is in posture p´, defender is in posture p´´; j = *poff*(p´´); i = *poff*(p´) if attacker is Red; i = vfeba.npa + *poff*(p´) if attacker is Blue	real
vfeba.f0(i,j)	signed distance of FEBA movement in 1 unit of time if attacker-to-defender force ratio is 0, attacker is in posture p´, and defender is in posture p´´; j = *poff*(p´´); i = *poff*(p´) if attacker is Red; i = vfeba.npa + *poff*(p´) if attacker is Blue	real
vfeba.fr(i)	ordinate corresponding to *vfeba.f*(j,k,i) for any (j,k) LB = 0	real
vfeba.npa	maximum number of attack postures subprogram vfeba can accommodate given current array dimensions	integer
vroll(i,j)	rate at which type i resources can move themselves on type j artery LB = 0	real
vsea(i)	rate at which type i resources can sail LB = 0	real

13. INDEX OF VARIABLES

14. REFERENCES

[1] Anderson, Lowell Bruce, *et al.* *IDA Ground-Air Model I (IDAGAM I).* Vol. I, R-199, Arlington, VA: Institute for Defense Analyses, October 1974.

[2] Anderson, Lowell Bruce. *A Method for Determing Linear Weighting Values for Individual Weapons Systems,* WP-4, Improved Methodologies for General Purpose Forces Planning (New Methods Study), Arlington, VA: Institute for Defense Analyses, Revised April 1973.

[3] Dare, D.P., and B.A.P. James. *The Derivation of Some Parameters for a Corps/Division Model from a Battle Group Model,* Defence Operational Analysis Establishment Memorandum 7120, U.K.: Ministry of Defence, July 1971, CONFIDENTIAL.

[4] Holter, William H., *et al.* *Appendix D: NATO Combat Capabilities Analysis II (COMCAP II)* (U), GRC Report OAD-CR-8, McLean, VA: General Research Corporation, August 1973, SECRET (Appendix D is UNCLASSIFIED).

[5] Howes, David R., and Robert M. Thrall. "A Theory of Ideal Linear Weights for Heterogeneous Combat Forces," *Naval Research Logistics Quarterly,* Vol. 20, No. 4, December 1973. (Or see Robert M. Thrall and Associates, Chapter 2C, *Final Report to US Army Strategy and Tactics Analysis Group* (RMT-200-R4-33), May 1972.)

[6] Spudich, John. *The Relative Kill Productivity Exchange Ratio Technique,* Combined Arms Research Office, Booz-Allen Applied Research, Inc., n.d.

[7] US Army Combat Developments Command, Headquarters, TAB E, Appendix II to Annex L, *Tank, Antitank and Assault Weapons Requirements Study* (U), Phase III (*TATAWS III*), December 1968. SECRET/NOFORN (Tab E, Appendix II to Annex L is UNCLASSIFIED).